Comprando esta edição tenha acesso na WEB a:
+440 vídeo-aulas +150 artigos e 6 cursos online

www.clubedelphi.net Feita para desenvolvedores

ClubeDelphi

Ano 7 · Edição 88 · R$9,90

DevMedia GROUP

ISSN 1517990-7

Mini-curso: Construa uma aplicação completa com o Delphi for PHP

Na Web

Confira um mini-curso completo sobre um dos componentes mais utilizados do Delphi, o **DBGrid**. Veja como personalizar o componente, suas colunas, ordenar, colocar cores, exibir imagens, campos Blob, calendários, CheckBoxes e muito mais.

Bugs?
Implemente um mecanismo eficiente para controlar bugs na aplicação e que possam ser gerenciados pela equipe de desenvolvimento

Mini-curso
Construa um sistema completo de Contas a Pagar e Cobrança com dbExpress e Firebird 2.0 – Parte 2

Delphi 7? 8? 2005? 2006? Turbo Delphi? Delphi 2007 for Win32?
Saiba as diferenças entre as versões e qual a melhor escolha para o seu projeto

Mão na Massa
Saiba como imprimir cheques com o Delphi

ASP.NET
Dicas para incrementar suas aplicações Web com JavaScript

Coluna Ask the Expert

Quantas cópias de seu software existem no mercado?

Esta é a chave mais segura do mundo contra pirataria de software.

Comprove você mesmo!

SafeNet®
The Foundation of Information Security

Alameda Tocantins, 280 - Barueri-SP

Tel: +55 11 4208-7700

http://br.safenet-inc.com - Contato: infobrasil@safenet-inc.com

ClubeDelphi

Ano 7 - 88ª Edição - 2007 - ISSN 1517990-7 Impresso no Brasil

Corpo Editorial

Diretor Editorial e Administrativo
Gladstone Matos
gladstone@neoficio.com.br

Equipe Editorial
Adriano Santos, Bruno Lichot, Everson Volaco, Fabrício Desbessel, Gustavo Chaurais e Michael Benford

Editor de Arte
Vinicius O. Andrade
viniciusoandrade@gmail.com

Capa
Antonio Xavier
webdesigner@devmedia.com.br

Editor Geral
Guinther Pauli
guinther@devmedia.com.br

Editor Técnico
Luciano Pimenta
lucianopimenta@devmedia.com.br

Revisão
Rafael de Castro Santos
rafael@devmedia.com.br

Distribuição
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Carmelita Mulin
www.devmedia.com.br/central/default.asp
(21) 2220-5375

Kaline Dolabella
Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 2220-5375

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site entre em contato com:

Kaline Dolabella
publicidade@devmedia.com.br

Para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

Jeff Wendell
jeff@devmedia.com.br

ClubeDelphi PLUS

DevMedia GROUP

A revista ClubeDelphi é parte integrante da assinatura ClubeDelphi PLUS. Para mais informações sobre o pacote PLUS, acesse: *http://www.devmedia.com.br/clubedelphi/portal.asp*

Apoio

ASSESPRO PARANÁ

ÍNDICE

06 - Delphi for PHP - Parte 1
Fabrício Desbessel

14 - Bug Tracking
Felipe La Rocca Teixeira

24 - Contas a Pagar e Cobrança - Parte 2
Adriano Santos

32 - Impressão de cheques
Adriano Santos

38 - Versões de Delphi
Fabrício Desbessel

44 - JavaScript
Luciano Pimenta

EDITORIAL

Com o lançamento do Delphi 8, os desenvolvedores Delphi tiverem o primeiro contato com o .NET (na verdade o D7 já trazia um compilador). O que mais me surpreendeu foi que o Delphi for .NET era amplamente esperado, mas parece que acabou sendo pouco utilizado. O .NET pareceu mais uma barreira para os desenvolvedores do que uma nova e promissora tecnologia. Todos sabiam que um dia teriam que fazer algo em .NET, mas parece que esse dia nunca chega... O Delphi 2007 trouxe novamente o que todos já sabiam, o desenvolvimento com o Delphi sempre vangloriou o Win32. Nesse meio tempo, vi muitos desenvolvedores migrando para o Java, devido a incerteza perante o uso do .NET.

Nesse contexto, surge uma terceira e nova opção: o Delphi for PHP. Significa muito mais do que simplesmente a quinta alternativa para o desenvolvimento Web com o Delphi, trata-se de uma linguagem já amplamente aceita e difundida mundialmente aliada a um IDE produtivo e robusto já bastante conhecido dos Delphianos. É a união dos melhor de dois grandes mundos, com um toque mágico: a VCL Web for PHP.

E nada melhor do que um mini-curso passo a passo, construindo uma aplicação completa, para aprender na prática esse ousado e promissor lançamento da CodeGear, nova divisão da Borland focada exclusivamente na produtividade do desenvolvedor. Nesta edição, Fabrício Desbessel, desenvolvedor com larga experiência em PHP e Delphi, introduz um excelente mini-curso, onde você poderá aprender tudo sobre o Delphi for PHP.

Ainda nesta edição, aprenda a gerenciar de forma controlada e organizada os bugs que acontecem na sua aplicação, através de um eficiente mecanismo de Bug Tracking. Quem nunca usou papéis com OSs (ordens de serviço) para controlar bugs e solicitações de cliente? Faça isso de forma eficaz a partir de agora. O Adriano continua o mini-curso que mostra como criar uma aplicação completa de contas a pagar e cobrança. Na coluna mão na massa, Adriano apresenta a impressão de cheques com Delphi.

Confuso com tantas versões do Delphi lançadas atualmente? Então leia o artigo do Fabrício e tire todas as suas dúvidas. E para finalizar, Luciano Pimenta dá boas dicas de uso de JavaScript para incrementar suas aplicações Web com ASP.NET.

Sucesso a todos e boa leitura!

Guinther Pauli
guinther@devmedia.com.br
Microsoft Certified: MCP, MCAD, MCSD.NET
Borland Certified: Delphi 6, 7, 2005, 2006, Web, Kylix

Fale com o Editor

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!

Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Guinther Pauli - Editor da Revista
guinther@devmedia.com.br

Luciano Pimenta - Editor do Site
lucianopimenta@devmedia.com.br

Portal do Assinante

NÃO PERCA

A ClubeDelphi tem uma novidade para você que comprou este exemplar na banca de jornal: você pode acessar GRATUITAMENTE, o Portal do Assinante ClubeDelphi!

Confira o que você encontra no Portal do Assinante:
- Mais de 400 Vídeo Aulas!
- 6 cursos online!
- 1 Livro Eletrônico sobre ADO.NET e BDP!
- Mais de 140 Artigos Exclusivos!

Para Utilizar o Portal do Assinante, acesse *www.devmedia.com.br/clubedelphi/portal.asp* e utilize as informações abaixo: **Login: DVM.PL e Senha: REV88**

O acesso é válido por 30 dias a partida da data de lançamento da revista. Todos os meses a ClubeDelphi lhe dará uma senha válida para acessar o portal. Comprando a revista regularmente em bancas, você terá acesso ininterrupto a ele!

Ask The Expert

Perguntas e Respostas

Dúvidas respondidas por **Luciano Pimenta**
(envie as suas para pimenta@devmedia.com.br)

Banners em Menu

Olá pessoal da ClubeDelphi. Em primeiro lugar gostaria de parabenizá-los pela alta qualidade da revista. Tenho uma dúvida e gostaria de uma ajuda. Dia desses vi uma aplicação que tinha um banner ao lado de um menu, estilo o menu Iniciar do Windows. Como eu faço isso em Delphi?

Jean Minussi

Para isso, adicione no formulário um *Image* e carregue uma imagem de sua preferência. Selecione todos os itens do menu e no evento *OnDrawItem* digite o código mostrado na **listagem 1**.

Rode e teste a aplicação (**Figura 1**). A dica mostrada com o *MainMenu* se estende ao *PopupMenu*.

Listagem 1. Código do evento OnDrawIten

```
procedure TForm1.Exit1DrawItem(Sender: TObject;
  ACanvas: TCanvas; ARect: TRect; Selected: Boolean);
begin
  ACanvas.FillRect(ARect);
  ACanvas.TextOut(ARect.Left + 48, ARect.Top,
    StripHotkey((Sender as TMenuItem).Caption));
  ACanvas.Draw(0, 0, Image1.Picture.Graphic);
  // desenha o ícone do menu
  ImageList1.Draw(ACanvas, ARect.Left + 30, ARect.Top,
    (Sender as TMenuItem).ImageIndex);
end;
```

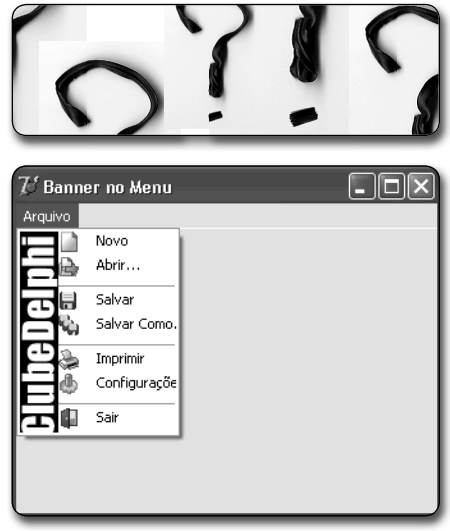


Figura 1. Banner no menu da aplicação

“ O Maior Portal para Desenvolvedores da América Latina.”

Acesse:

www.devmedia.com.br

DevMedia GROUP

Assinatura
ClubeDelphi PLUS
Mais conteúdo .NET por menos!

+400 vídeo aulas | 6 cursos online

www.clubedelphi.net/portal

Caro Leitor,

O portal ClubeDelphi PLUS é a continuação, na Web, da revista ClubeDelphi. O portal recebe um conteúdo novo todo dia e hoje conta com: i) mais de 400 vídeo aulas; ii) 6 cursos online; iii) 1 livro eletrônico gratuito, de Guinther Pauli, sobre ADO.NET e BDP; iv) mais de 150 artigos exclusivos (que não foram publicados na revista)!;
Acesse o portal ClubeDelphi PLUS e receba muito mais conteúdo sobre Delphi! E o que é melhor: de graça! Todo leitor da revista ClubeDelphi, seja ele assinante ou comprador da revista em bancas, tem acesso ao portal (para quem compra em bancas, o acesso é válido por 30 dias).
Se você é assinante, utilize o seu login e senha pessoais para acessar o portal. Se você comprou em bancas, utilize o login e senha publicados na página do editorial desta edição.
Confira a seguir as últimas novidades do portal!

Boa leitura e sucesso!
Equipe DevMedia

Curso Online - PLUS

O portal ClubeDelphi PLUS conta com 6 cursos online, confira!

Sistema completo com Delphi 7, dbExpress e Firebird 2.0 [*Luciano Pimenta*]

Aplicações WEB com IW e Delphi 7 (Delphi Win32)! [*Guinther Pauli*]

Aplicações client/server no Delphi 2006 [*Luciano Pimenta*]

Criando uma Aplicação multi-camadas Completa com Delphi [*Guinther Pauli*]

Criando uma aplicação completa: sistema SysCash [*Everson Volaco*]

Aplicações Client/Server com dbExpress e Firebird [*Luciano Pimenta*]

.:: DESTAQUE ::.

Confira no portal ClubeDelphi PLUS vídeo aulas sobre dicas avançadas de DBGrid

Mini curso de ClientDataSet
www.devmedia.com.br/articles/listcomp.asp?txtsearch=DBGrid

Últimas Vídeo-Aulas (Somente para assinantes)

Sistema completo com Delphi 7, dbExpress e Firebird 2.0
Acompanhe a criação de um sistema completo de gerenciamento de locadora, usando Delphi, dbExpress e Firebird 2.0. Luciano Pimenta, mostrará todos os passos necessários para a criação do projeto em todas as fases de implementação, desde a criação do banco de dados, até a migração para .NET.

Mini-curso de ClientDataSet - Parte XVII a XX
Veja nessas aulas de Guinther Pauli um mini-curso exclusivo sobre dicas avançadas de ClientDataSet.
Componente Window (Janela) no Delphi for PHP
Veja nessa vídeo aula de Fabricio Desbessel, como trabalhar com o componente Window no Delphi for PHP.

Apresentando a suíte UserControls - Parte III a V
Veja nessa vídeo aula de Paulo Quicoli a suíte User Controls.

Segredos do DBGrid – Parte I a IV
Veja nessas vídeo aulas de Guinther Pauli, várias dicas avançadas de uns dos componentes mais usados da VCL, o DBGrid.

Conhecendo o JEDI VCS Client
Veja nessa vídeo aula de Adriano Santos, como utilizar o JEDI VCS Client.

Conhecendo o JEDI VCS Server
Veja nessa vídeo aula de Adriano Santos, como utilizar o JEDI VCS Server.

Mais conteúdo para um Treeview
Veja nessa vídeo aula de Paulo Quicoli como criar um Treeview turbinado.

Delphi for PHP - Frames
Veja nessa vídeo aula de Adriano Santos, como trabalhar com Frames no Delphi for PHP.
Autenticação no Delphi for PHP - Parte I a IV
Veja nessa aula de Fabrício Desbessel como criar autenticação no Delphi for PHP.

Crie um loja virtual com Delphi for PHP
Veja nessa vídeo aula de Fabricio Desbessel, o inicio de mais um curso on-line do Portal ClubeDelphi. Aprenda a criar uma loja virtual com Delphi for PHP.

Delphi for PHP - Parte 1

Crie um site de publicação de notícias

Fabrício Desbessel

(fabricio@fabricio.pro.br)

é professor de Linguagem de Programação do Curso Técnico em Informática do Colégio Frederico Jorge Logemann de Horizontina/RS e da FAHOR Faculdade Horizontina. Delphiano de coração está sempre disposto a provar que com o Delphi sempre teremos a melhor solução. Site www.fabricio.pro.br.

Estou muito feliz, pois poderei compartilhar com os leitores da ClubeDelphi, a partir dessa edição, alguns conhecimentos, adquiridos em horas de estudo e prática com essa ótima ferramenta. Confesso que me diverti bastante, pois é muito prazeroso aprender a desenvolver com o Delphi for PHP e, relembrei como também era divertido aprender Delphi 1, 2, 3, 4, 5, 6 e 7.

Nesta série de artigos vamos criar um sistema de publicação de notícias, utilizando o banco de dados MySQL. O objetivo é mostrar como criar uma aplicação para a manutenção de dados, com controle de acesso administrativo e páginas para os visitantes.

Nesse primeiro artigo vamos criar o banco de dados, fazer o registro do banco no Delphi for PHP, adicionar os componentes de acesso ao banco, bem como criar os formulários para inserção, alteração e exclusão dos registros.

Criando o Banco de dados

Para a criação do banco de dados você deve ter instalado o MySQL Server 5.0 (baixe em *dev.mysql.com*) e o MySQL Administrador (*dev.mysql.com/downloads/gui-tools/index.html*). A estrutura do banco está representada no E.R, exibido na **Figura 1**.

Para criar o banco de dados abra o *MySQL Administrator*, e faça o *login* em *localhost*. Clique em *Catalogs* e em *Schema* clique com o botão direito e escolha *Create New Schema* (**Figura 2**). Você também pode utilizar o atalho das teclas CTRL+N. No diálogo *Create new Schema*, informe "jornal" para o campo *Schema name*. Após a criação do banco de dados devemos executar o *script* SQL para criação das ta-

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora mesmo o portal do assinante ClubeDelphi e assista as aulas de Fabrício Desbessel que ensinam a instalar o banco de dados MySQL Server 5.0 e as ferramentas de administração.

www.devmedia.com.br/articles/viewcomp.asp?comp=4985

belas e chaves. Para tanto, acesse o menu *Tools>MySQL Command Line Client*. Digite: "use jornal;" e pressione ENTER.

Copie o código da **Listagem 1** e cole no *command*. Finalize pressionando ENTER novamente, para que as tabelas sejam criadas.

Você pode verificar a criação das tabelas, digitando o comando "show tables;", que exibe a lista de tabelas existentes no banco. Para verificar a estrutura de uma tabela utilize o comando *desc* seguido do nome da mesma, como, por exemplo, "desc noticias;". Lembre-se que todos os comandos devem ser finalizados com ponto e vírgula (;) para serem executados. Após finalizar o processo de criação digite "quit" para sair do *command*.

Conectando o Delphi for PHP ao MySQL

Abra o Delphi for PHP e acesse o *Data Explorer*. Clique no sinal de mais (+) para abrir os tipos de conexões padrões. Clique com o botão direito em *MySQL* e selecione a opção *Register Database*.

Na tela de registro informe "bd_jornal" para o campo *Connection Name*. No campo *Host* informe "localhost" que é a própria máquina (ou indique um servidor do MySQL que esteja utilizando). Para autenticação deve-se informar o usuário e senha do MySQL que podem acessar o banco que criamos anteriormente.

No campo *Database* informe "jornal", que foi o nome que utilizamos na criação do banco. Na **Figura 3** podemos verificar os campos preenchidos da maneira correta.

Iniciando a aplicação

No Delphi for PHP acesse o menu *File>New>Application*. No formulário que será exibido, altere sua propriedade *Name* para "UPrincipal". Para salvar o projeto devemos criar um diretório, de preferência no C:\ para que não haja problemas (Update 1 resolveu esse bug, que ocorria quando o projeto era salvo em um *drive* diferente de onde estava instalado Delphi for PHP).

Clique no botão *Save All* (SHIFT+CTRL+S) e informe o nome de "principal.php" para a unit e "PJornal.phprj" para o projeto. Tome muito cuidado com a nomenclatura das *units*, pois serão os nomes das páginas PHP. Evite utilizar acentuações, espaços e letras maiúsculas.

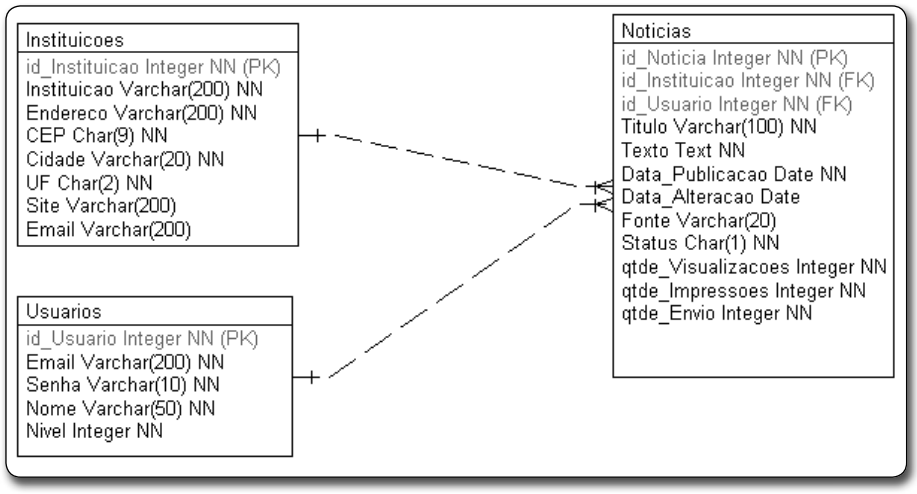


Figura 1. Modelo E.R. da aplicação

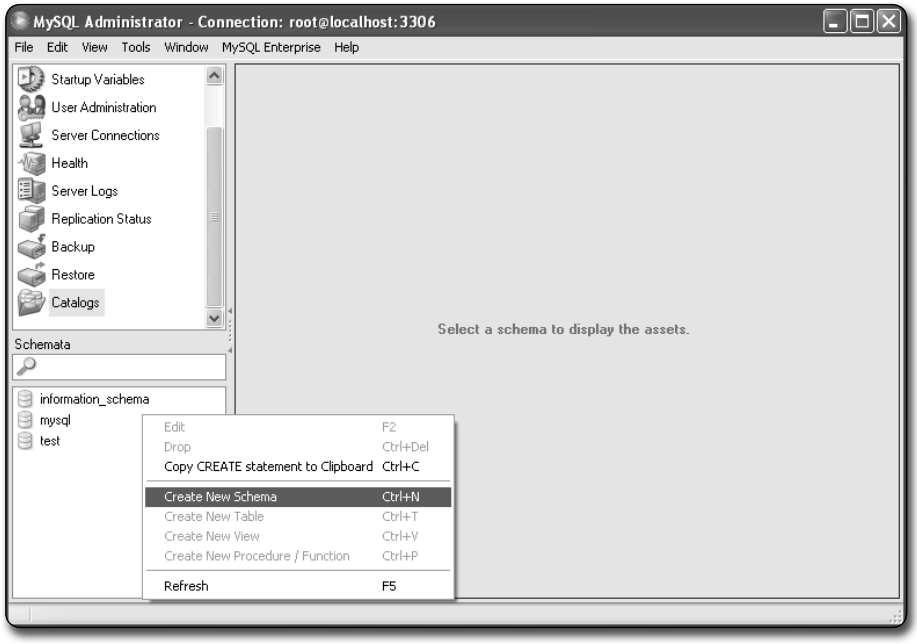


Figura 2. Criação do novo banco no MySQL Administrator

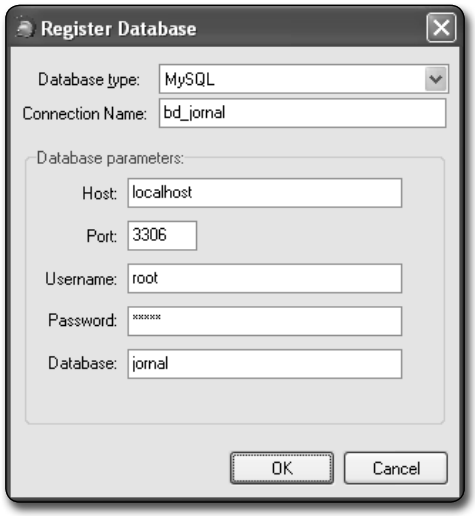


Figura 3. Registrando o banco de dados

Componentes de acesso a dados

Para colocar os componentes de acesso a dados vamos utilizar nosso velho conhecido Data Module. Para criá-lo acesse o menu *File>New>Data Module*. Altere a propriedade *Name* para "UDados" e salve o arquivo como "dados.php".

Agora bastará arrastar as tabelas do *Data Explorer* para o Data Module, para que os componentes necessários sejam criados automaticamente. Para fazer isso, clique ao lado da conexão do MySQL com o nome de *bd_jornal* e depois em *Tables*.

Pegue cada tabela existente e arraste para o Data Module. É importante fazer uma organização visual para entendermos as ligações dos componentes (**Figura 4**).

Todos os componentes adicionados são seguidos dos nomes e de um numeral. O *Database* faz a ligação com o servidor do banco de dados, e o *Table* representa uma tabela e possui todos os métodos para manutenção dos registros. O *DataSource* faz a preparação para apresentação dos dados. Sentiram alguma semelhança com o Delphi 7 / Win32?

Formulários para manutenção dos registros

Crie um novo formulário através do menu *File>New>Form*. Mude a propriedade *Name* para "UInstituicoes" e salve o arquivo como "instituicoes.php". Como nossos componentes de acesso ao banco estão no Data Module, vamos fazer a referência através do menu *File>Use Unit* (ALT+F11), escolhendo *dados.php*.

Primeiramente vamos colocar os botões que possibilitaram a manutenção dos dados. Adicione seis *Buttons* no topo do formulário e configure-os conforme a **Tabela 1**.

A idéia de deixar os botões *Gravar* e *Cancelar* desabilitados é impedir a execução desses códigos se o usuário não pressionar antes o *Novo* ou *Alterar*, ou seja, é necessário inserir ou editar um registro para poder salvar ou cancelar, claro. Após colocar os botões verifique a propriedade *Top* de cada um para saber se estão com a mesma posição.

Agora coloque um *Label* para cada campo da tabela *instituicoes*, alterando sua propriedade *Caption* para o nome do campo, devidamente correto ortograficamente. Deixe espaços entre os *Labels* para a inserção do próximo componente.

Vamos colocar entre os *Labels*, os *Edits* que estarão ligados à tabela e a cada campo em específico. Adicione um *Edit* para cada campo da tabela e altere as propriedades *DataSource* para *UDados*. *dsinstituicoes1* e *DataField* para o nome do campo na tabela, que deve ser digitado.

> **Nota:** Cuidado ao digitar a propriedade *DataField* porque o PHP diferencia letras maiúsculas das minúsculas. Coloque realmente igual ao campo da tabela.

Para o UF podemos utilizar um *ComboBox* ao invés do *Edit*. Vamos fazer essa troca, excluindo o *Edit* do campo UF e colocando um *ComboBox* no lugar. Configure também as propriedade *DataSource* e *DataField* como feito anteriormente para o *Edit*.

Clique na propriedade *Items* para co-

Listagem 1. Script para criação das tabelas

```
create table instituicoes (
  id_instituicao int not null auto_increment
    comment 'identificador da instituição', instituicao varchar(200) not null
    comment 'nome completo', endereco varchar(200) not null
    comment 'endereço completo', cep char(9) not null
    comment 'código de endereçamento postal', cidade varchar(20) not null
    comment 'nome da cidade', uf char(2) not null
    comment 'unidade da federação (sigla do estado)', site varchar(200)
    comment 'url', email varchar(200)
    comment 'endereço eletrônico',  primary key (id_instituicao))
    comment = 'instituições que publicam notícias';

create table usuarios (
  id_usuario int not null auto_increment
    comment 'identificador do usuário', email varchar(200) not null
    comment 'endereço eletrônico que servirá de login', senha varchar(10) not null
    comment 'informação secreta', nome varchar(50) not null
    comment 'nome completo', nivel int not null
    comment 'nível de acesso ao sistema', primary key (id_usuario))
    comment = 'controle de usuários que podem acessar o sistema';

create table noticias (
  id_noticia int not null auto_increment comment 'identificador da notícia',
  id_instituicao int not null,
  id_usuario int not null,
  titulo varchar(100) not null comment 'título principal',
  texto text not null comment 'corpo da notícia',
  data_publicacao date not null comment 'data de edição da notícia',
  data_alteracao date comment 'data de alteração (se houver)',
  fonte varchar(20) comment 'url da fonte se for notícia externa',
  status char(1) not null default 'a' comment 'status da notícia valores:
     a - aguardando  p - publicada  c - cancelada',
  qtde_visualizacoes int not null default 0 comment 'contador de visualizações',
  qtde_impressoes int not null default 0 comment 'contador de impressões',
  qtde_envio int not null default 0 comment 'contador de envios por e-mail',
  primary key (id_noticia))  comment = 'notícias publicadas';

alter table noticias add foreign key (id_instituicao)
references instituicoes (id_instituicao)
on delete restrict on update restrict;

alter table noticias add foreign key (id_usuario)
references usuarios (id_usuario)
on delete restrict on update restrict;
```

Ord/P	Caption	Enabled	Hint	Name	ShowHint
1	Novo	True	Inserir	btnNovo	True
2	Alterar	True	Editar	btnAlterar	True
3	Gravar	False	Salvar	btnGravar	True
4	Cancelar	False	Desistir	btnCancelar	True
5	Deletar	True	Excluir	btnDeletar	True
6	Voltar	True	Retornar	btnVoltar	True

Tabela 1. Propriedade dos Buttons da aplicação

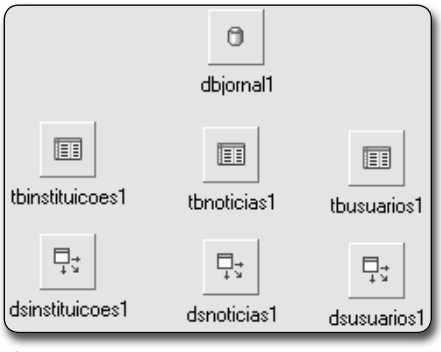


Figura 4. Data Module organizado

locar as opções que aparecerão na lista. Basta informar uma chave e um valor. Por exemplo, informe RS tanto para *Key* como para *Value*. Clique em *Add* e adicione também um para PR (**Figura 5**).

Com a interface pronta, como mostra a **Figura 5**, devemos codificar os botões seguindo a ordem da disposição no formulário e, acessando o evento *OnClick*, pelo *Object Inspector* ou através de um clique duplo sobre cada botão, usando o código da **Listagem 2** para o *Novo*.

Explicando o código da **Listagem 2**, primeiramente, precisamos utilizar o comando *global* do PHP que ordena que a variável seja procurada fora da própria função. Precisamos fazer isso, pois a variável *$UDados* está presente na *unit* local (isso porque executamos um *File>Use Unit*), mas está fora da função.

Em segundo lugar colocamos um filtro na tabela (0) que não trará nenhum registro, mas dará um *Open* na mesma, o que é necessário pela ligação dos *Edits* aos *DataFields* (com a tabela fechada acontece erros). O comando *Refresh* atualiza a consulta, ou seja, executa novamente utilizando o filtro.

O comando de inserção utilizado é o *Append* que insere um registro após o último existente no banco, também podemos utilizar o *Insert*. Habilitamos e desabilitamos alguns botões com o intuito de deixar disponível ao usuário somente o que ele pode fazer.

Por último, utilizamos nossa conhecida propriedade *Tag* para guardar a informação que estamos fazendo uma inserção. Para o botão *Alterar*, use o código da **Listagem 3**.

No código anterior, temos o comando para alterar um registro, que é o *Edit*. No último comando, passamos o valor 2 para o propriedade *Tag* com o objetivo de informar que é uma alteração. Essa informação de inserção ou alteração será necessária nos próximos códigos.

No código da **Listagem 4** temos o *Post* que grava as informações no banco de dados. Para saber como deve ser apresentada a página, depois de gravar as informações, utilizamos a propriedade *Tag* do *btnGravar*. A idéia é que, se for uma inserção, exiba os dados inseridos e atualize o código que é auto-incremento.

Para fazer isso, utilizamos uma função do PHP que traz o último valor para um campo auto-incremento (*mysql_insert_id()*), onde utilizamos em nosso filtro para trazer o registro inserido. Se for uma alteração a página permanece com os dados alterados e por isso não precisamos tratar.

Na **Listagem 5** temos o *Cancel* que trata de cancelar a inserção ou edição de um registro. Se cancelarmos uma inserção, retornaremos a página com os campos em branco e fazemos isso passando um filtro zerado (0).

Já se cancelamos uma edição, retornamos uma página com o registro que estava sendo alterado, mas com os valores antigos, ou seja, sem salvar a alteração. Para o *Deletar*, use o código da **Listagem 6**.

O comando para deletar, obviamente é o *Delete* como o apresentado na **Listagem 6**, que também limpa o filtro e apresenta o primeiro registro da consulta. Para o *Voltar*, vamos redirecionar a aplicação para a página principal do sistema, usando o seguinte código:

```
redirect('principal.php');
```

Bem, nesse momento já podemos testar nossa aplicação. Faça os testes de inserção, alteração e exclusão de registros (**Figura 6**).

Listagem 2. Código do botão Novo

```
//Transformar a classe em variável local
global $UDados;
//Abrir um registro em branco
$UDados->tbinstituicoes1->Filter="id_instituicao=0";
$UDados->tbinstituicoes1->Refresh();
//Dar o comando de Inserção
$UDados->tbinstituicoes1->Append();
//Habilitar e Desabilitar os botões
$this->btnNovo->Enabled=false;
$this->btnAlterar->Enabled=false;
$this->btnGravar->Enabled=true;
$this->btnCancelar->Enabled=true;
$this->btnDeletar->Enabled=false;
$this->btnVoltar->Enabled=false;
//Passamos 1 para informar que é inserção
$this->btnGravar->Tag=1;
```

Listagem 3. Código do botão Alterar

```
//Transformar a classe em variável local
global $UDados;
//Dar o comando de Alterar
$UDados->tbinstituicoes1->Edit();
//Habilitar e Desabilitar os botões
$this->btnNovo->Enabled=false;
$this->btnAlterar->Enabled=false;
$this->btnGravar->Enabled=true;
$this->btnCancelar->Enabled=true;
$this->btnDeletar->Enabled=false;
$this->btnVoltar->Enabled=false;
//Passamos 2 para informar que é alteração
$this->btnGravar->Tag=2;
```

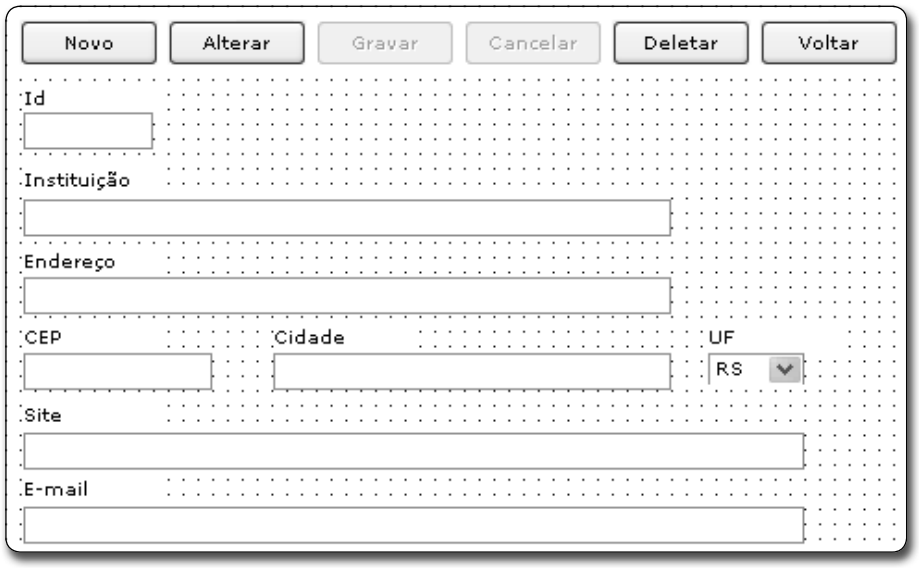


Figura 5. Layout do formulário montado

Listagem 4. Código do botão Gravar

```
//Transformar a classe em variável local
global $UDados;
//Dar o comando de Gravar
$UDados->tbinstituicoes1->Post();
//Dar o comando de Atualizar e localizar o último registro
//Se for uma inserção
if ($this->btnGravar->Tag==1)
{
  $UDados->tbinstituicoes1->
    Filter="id_instituicao=".mysql_insert_id();
  $UDados->tbinstituicoes1->Refresh();
}
//Habilitar e Desabilitar os botões
$this->btnNovo->Enabled=true;
$this->btnAlterar->Enabled=true;
$this->btnGravar->Enabled=false;
$this->btnCancelar->Enabled=false;
$this->btnDeletar->Enabled=true;
$this->btnVoltar->Enabled=true;
```

Listagem 5. Código do botão Cancelar

```
//Transformar a classe em variável local
global $UDados;
//Dar o comando de Cancelar
$UDados->tbinstituicoes1->Cancel();
//Se for uma inserção deixar campos em branco
if ($this->btnGravar->Tag==1)
{
  $UDados->tbinstituicoes1->Filter="id_instituicao=0";
} else
//Se for alteração passar para o filtro o registro que
//estava sendo alterado
if ($this->btnGravar->Tag==2)
{
  $UDados->tbinstituicoes1->
    Filter="id_instituicao=".$this->Edit1->Text;
  //Atualizar a consulta
  $UDados->tbinstituicoes1->Refresh();
  //Habilitar e Desabilitar os botões
  $this->btnNovo->Enabled=true;
  $this->btnAlterar->Enabled=true;
  $this->btnGravar->Enabled=false;
  $this->btnCancelar->Enabled=false;
  $this->btnDeletar->Enabled=true;
  $this->btnVoltar->Enabled=true;
}
```

Listagem 6. Código do botão Deletar

```
//Transformar a classe em variável local
global $UDados;
//Dar o comando de Deletar
$UDados->tbinstituicoes1->Delete();
//Limpar o Filtro
$UDados->tbinstituicoes1->Filter="";
//Atualizar
$UDados->tbinstituicoes1->Refresh();
```

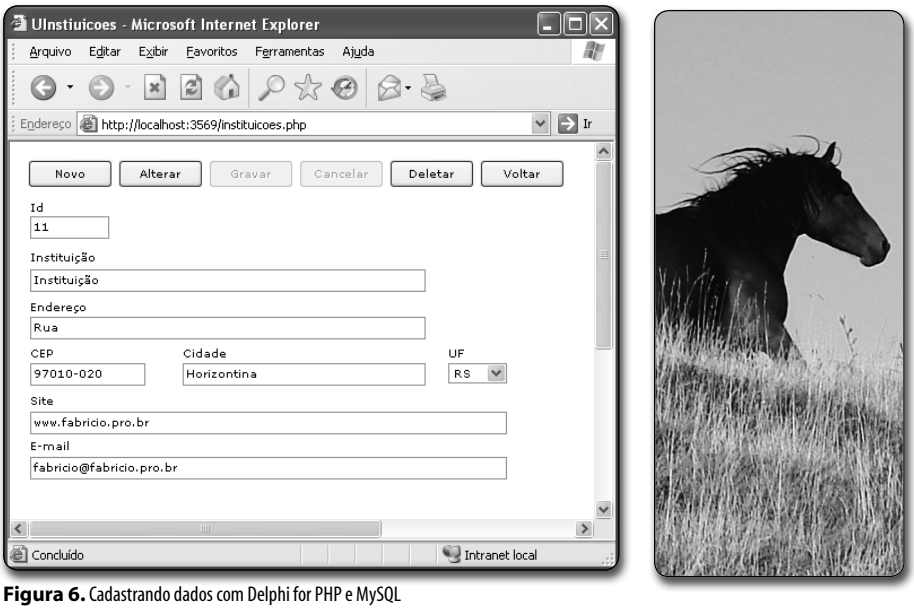


Figura 6. Cadastrando dados com Delphi for PHP e MySQL

Validação

A validação é um passo muito simples no Delphi for PHP, pois, o *Button* possui um evento *JavaScript* do tipo *OnClick*. Então, selecione o *btnGravar* e no *Object Inspector*, na aba *JavaScript* acesse o evento *OnClick*. Codifique-o conforme a **Listagem 7**. Como trata-se de um código *JavaScript* tome o cuidado de colocar o código após o comentário *"//Add your javascript code here"*, pois esse local está fora do código PHP.

Para quem já conhece *JavaScript* o entendimento da **Listagem 7** é rápido. Para quem não conhece explicarei: para os campos obrigatórios testamos se estamos no campo, informando o nome dos *Edits* e depois verificando se tem valor, ou seja, o campo foi preenchido.

Para o nome dos campos, olha-se a propriedade *Name* do *Edit*. Em campos que possuem um tamanho determinado, como o CEP, verificamos o tamanho do valor preenchido através da propriedade *Length* (utilizei nove caracteres pois uso o "-" no CEP). Então basta ter um *if* para cada campo que deve ser preenchido.

Como temos a mudança da variável *result* para *false*, caso o campo não esteja preenchido, testamos, no fim do código, se deve ser exibida a mensagem ou não. Também a cada falta de digitação adicionamos uma linha de texto na variável *msg* explicando ao usuário o que deve ser feito. Se o *return* for igual a *false*, a página não será submetida ao servidor.

Faça você mesmo

Agora chegou a hora de você colocar a mão na massa. Repita os passos anteriores para criar um formulário para a tabela *usuarios*. Lembre-se de primeiramente criar o formulário, depois colocar os campos, botões e fazer todas as codificações. Para agilizar o processo, utilize o "arrastar e soltar" para os campos da tabela.

O inconveniente disso é que o Delphi for PHP não reconhece a tabela existente no Data Module e cria todos os componentes de acesso no próprio formulário. Podemos corrigir isso excluindo os componentes e alterando a propriedade *DataSource* de todos os *Edits* para o *DataSource* existente em *UIDados* (*dados.php*).

Claro que para isso, não podemos esquecer de adicionar a unit como des-

crito anteriormente. Antes de arrastar os campos deve ser feita a seleção do componente que será criado pelo Delphi for PHP. Para tanto, na parte superior do *Data Explorer* temos alguns botões.

Os dois últimos tratam do componente para campos. Clique no último botão (*Edit*) que ficará selecionado e informará ao Delphi for PHP para criar *Edits*. Agora basta achar os campos no *Data Explorer* e arrastar para o formulário.

Quanto aos códigos dos botões, será necessário somente alterar o nome do *Table* em questão, bem como o filtro (*Filter*), levando em conta o campo chave da tabela em questão. Ao final o formulário para cadastro de usuários, deverá ter a mesma apresentação da **Figura 7**.

Para o campo *senha*, altere a propriedade *IsPassword* do *Edit* para *True*. Para o campo *Nível* utilizou-se um *ComboBox*, especificando-se os Itens ("1", "2", "3") que serão os níveis de acesso ao nosso sistema.

Siga os mesmos passos para, também criar um formulário para manutenção de notícias, que deverá ter a mesma aparência da **Figura 8**. Para esse formulário, utilizou-se o *Memo* para o campo *texto*, e alterei as propriedade *DataSource* e *DataField*.

Os campos de estatísticas (visualização, impressão e envio), mais o campo *id*, devem estar com a propriedade *ReadOnly=true* para que não seja permitida a alteração. O campo do usuário, vamos manter permitindo para os primeiros testes, pois, depois que tivermos a autenticação na aplicação não precisaremos mais.

Também utilizou-se um *ComboBox* para o campo *Instituição* (*id_instituicao*), que deverá apresentar a lista de instituições cadastradas no sistema. Para fazer isso, primeiramente altere o nome do componente

Listagem 7. Código do evento OnClick (JavaScript) do btnGravar

```
msg="";
result=true;
if ((document.UInstituicoes.Edit2) && (document.UInstituicoes.Edit2.value==""))
{
  result=false;
  msg+="-Preencha o Nome da Instituição\n";
}
if ((document.UInstituicoes.Edit3) && (document.UInstituicoes.Edit3.value==""))
{
  result=false;
  msg+="-Preencha o Endereço\n";
}
if ((document.UInstituicoes.Edit4) && (document.UInstituicoes.Edit4.value==""))
{
  result=false;
  msg+="-Preencha o CEP\n";
}
else
{
  if ((document.UInstituicoes.Edit4) &&
  (document.UInstituicoes.Edit4.value.length!=9))
  {
    result=false;
    msg+="-CEP Incorreto\n";
  }
}
if ((document.UInstituicoes.Edit5) && (document.UInstituicoes.Edit5.value==""))
{
  result=false;
  msg+="-Preencha a Cidade\n";
}
if ((document.UInstituicoes.Edit6) && (document.UInstituicoes.Edit6.value==""))
{
  result=false;
  msg+="-Escolha o UF\n";
}
if (!result)
{
  alert(msg);
}
return(result);
```

para "cbxInstituicoes".

Adicione ao formulário um *Query* da paleta *DataAccess*. Ligue sua propriedade *DataBase* para *UDados.dbjornal1* e na propriedade SQL informe a instrução SQL a seguir:

```
select id_instituicao, instituicao
from instituicoes
order by instituicao
```

Mude o *Name* para "sqlInstituicoes" e ative a SQL através da propriedade *Active* informando *true*. Vamos utilizar a consulta resultante para preencher os itens do *cbxInstituicoes*. No evento *OnBeforeShow* do *cbxInstituicoes* coloque o código contido na **Listagem 8**.

No código da **Listagem 8**, utilizamos o *AddItem* onde passamos, primeiramente o nome da instituição, depois um valor nulo e, por último o código da instituição que será a chave (*key*) do item.

Para finalizar, precisamos alterar o código do *Gravar*, que será diferente do primeiro exemplo. Essa alteração se faz necessária para passarmos a data para o campo data de publicação, bem como buscar os itens selecionados nos *ComboBox*, passando o chave (*key*) ao invés do valor (*value*).

Para saber os valores a serem salvos/exibidos no *Status*, veja o comentário do campo na **Listagem 1**. Então acesse o código do *Gravar* e, depois da declaração *global* e antes do *Post* adicione o código da **Listagem 9**.

No código anterior verificamos como é prático buscar ou atribuir um valor para o campo de um *DataSet*. Para pegarmos a data atual usamos o *date* que, no PHP exige um parâmetro de formatação da data, onde preferiu-se informar no mesmo formato que o MySQL utiliza (ano-mês-dia). Agora compile o formulário de notícias e teste.

Conclusão

A idéia desse artigo é mostrar como trabalhar no Delphi for PHP utilizando muitos conhecimentos do Delphi 7. Por isso, não estamos nos preocupando com a aparência, mas sim com o mesmo funcionamento de nossos sistemas desktop. Ao concluir esse artigo têm-se todos os cadastros funcionais e, no próximo vamos criar formulários para pesquisa. Coloque a mão na massa e crie esse exemplo no Delphi for PHP. ●

Listagem 8. Código do evento OnBeforeShow do cbxInstituicoes

```
while (!$this->sqlInstituicoes->EOF)
{
  $id = $this->sqlInstituicoes->id_instituicao;
  $instituicao = $this->sqlInstituicoes->instituicao;
  $this->cbxInstituicoes->AddItem(
    $instituicao, null, $id);
  $this->sqlInstituicoes->Next();
}
```

Listagem 9. Código adicionado ao Gravar

```
//Passar campos por código
$UDados->tbnoticias1->id_instituicao =
  $this->cbxInstituicoes->ItemIndex;
$UDados->tbnoticias1->status =
  $this->cbxStatus->ItemIndex;
$UDados->tbnoticias1->data_publicacao = date("y-m-d");
```

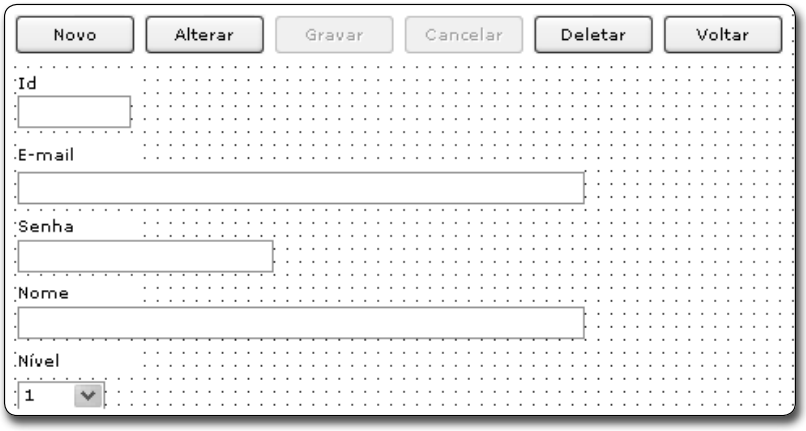


Figura 7. Formulário para manutenção de usuários

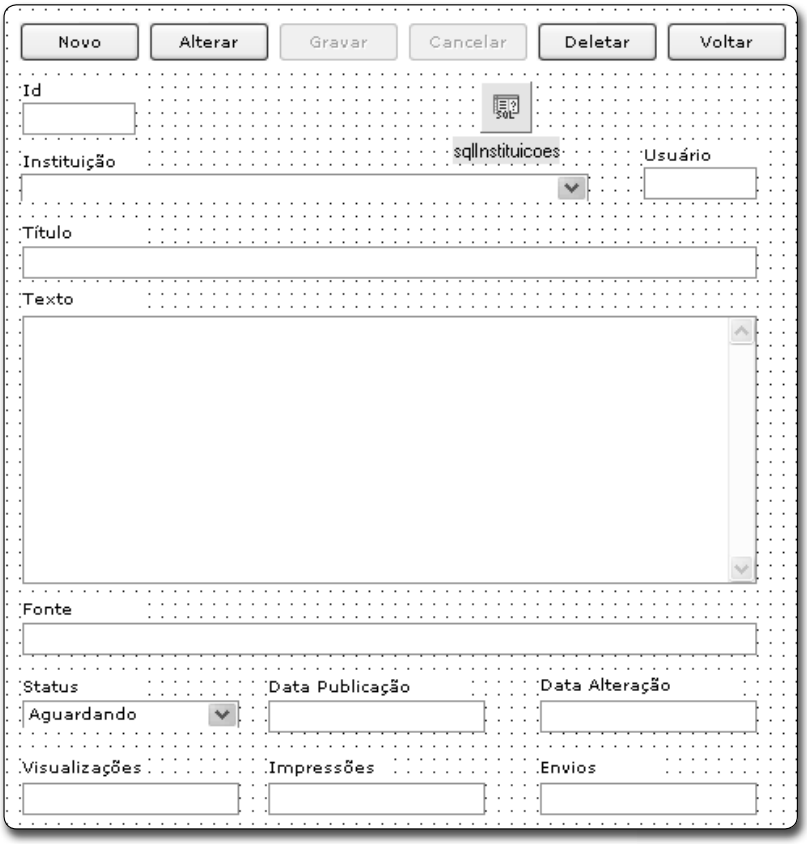


Figura 8. Formulário para manutenção de Notícias

NOVOS PLANOS

Descubra porque Hospedix é a hospedagem preferida dos profissionais.

Plano **Profissional** 1

1 Registro de domínio grátis*
3 Domínios por conta
50 GB de transferência
30 GB para e-mails com SSL
1 GB de espaço
ASP, ASP.NET 2.0 e PHP 5
Access e MySQL 5
SQL Server 2005 Express
Painel de controle e webmail
Editor de páginas on-line
Anti-spam e Anti-vírus

R$ 25,90 por mês

30 DIAS GRÁTIS PARA EXPERIMENTAR
Ao assinar, digite o código de desconto abaixo exclusivo para leitores da .NET Magazine e ganhe o primeiro mês inteiramente grátis.

RVSTNET

DOMÍNIO GRÁTIS ENQUANTO FOR CLIENTE
Só na Hospedix você tem registro de domínio (.com .net .org ou .info) grátis e isento de taxas anuais enquanto for cliente.

PAINEL DE CONTROLE PLESK 8.1 PROFISSIONAL
Painel de controle completo onde você mesmo adiciona domínios, sub-domínios, cria contas de e-mail, e muito mais.

Transfira seu site hoje mesmo para a Hospedix e descubra porque somos a empresa de hospedagem preferida dos profissionais.

www.hospedix.com.br

HOSPEDIX
HOSPEDAGEM PROFISSIONAL

As promoções anunciadas não são cumulativas com outras promoções | * Domínio .com .net .org ou .info grátis e isento de taxas anuais enquanto for cliente Hospedix.

Bug Tracking

Implementando Bug Tracking em uma aplicação apoiada por Controle de Mudanças

Felipe La Rocca Teixeira
(felipe.lt@pjf.mg.gov.br)
é Bacharel em Sistemas de Informação pelo Centro de Ensino Superior de Juiz de Fora (CES/JF) e Analista de Sistemas da Prefeitura de Juiz de Fora.

Marco Antônio Pereira Araújo
(maraujo@cesjf.br)
é professor do Curso de Bacharelado em Sistemas de Informação do Centro de Ensino Superior de Juiz de Fora (CES/JF), Doutorando e Mestre em Engenharia de Sistemas e Computação pela COPPE/UFRJ, Especialista em Métodos Estatísticos Computacionais e Bacharel em Informática pela UFJF, Analista de Sistemas da Prefeitura de Juiz de Fora.

Atualmente com o rápido avanço da tecnologia, desenvolver e manter sistemas de informação tornou-se algo difícil e complexo. Toda essa nova tecnologia também gera um aumento de novas solicitações e expectativas do usuário final e, para que possamos construir softwares de qualidade, é necessário não somente uma equipe técnica competente, mas também uma comunicação efetiva entre equipe de desenvolvimento e usuário final.

Ao longo do ciclo de vida de um software, modificações são inevitáveis, podendo ocorrer mudanças nas necessidades dos usuários, alterações do ambiente onde o sistema será utilizado, correções de defeitos, melhorias das funcionalidades do sistema.

Essas modificações afetam todos os tipos de artefatos como requisitos do sistema, documentos de análise e projeto, código-fonte e testes. A Gerência de Configuração é um conjunto de atividades projetadas para controlar mudanças inerentes ao desenvolvimento, mantendo a estabilidade na evolução do software, desempenhando um papel importante durante o desenvolvimento do projeto, tais como:

- Identificar os itens de configuração de acordo com as funcionalidades que deverão desempenhar;
- Documentar os itens ao longo do processo de desenvolvimento, gerando interação entre eles;
- Controlar as modificações ocorridas nos itens;
- Auditar modificações, garantindo a confiabilidade do produto.

Não é objetivo da gerência de configuração evitar modificações, mas permitir que elas ocorram de maneira controlada. Neste artigo será discutida uma de suas atividades principais, que é o controle de mudança.

Esse controle pode ser feito através do uso de ferramentas conhecidas como

Bug Tracking, permitindo assim ao desenvolvedor rastrear essas mudanças e identificar o impacto no projeto como um todo.

Controle de mudanças

É uma coleção de procedimentos documentados e formais que definem como as mudanças no software serão monitoradas e avaliadas. Oferece processos para identificar, analisar, rastrear e controlar mudanças. O uso de ferramentas possibilita a melhoria dos processos utilizados no desenvolvimento e manutenção de produtos de software.

As ferramentas de *Bug Tracking* estão prontas para auxiliar desenvolvedores no controle de mudanças, permitindo também que usuários façam o acompanhamento completo dos pedidos de alterações sobre erros encontrados no sistema. No mercado existem várias ferramentas *open source* de *Bug Tracking* disponíveis como *Bugzilla* e *Trac*.

Principais funcionalidades que uma ferramenta de Controle de Mudanças deve possuir:

- Acompanhamento de todo o ciclo de vida do pedido de mudança;
- Anexar arquivos ao pedido para facilitar o entendimento do problema ou complementar a especificação;
- Configuração do fluxo de trabalho, definindo os estados pelo qual um pedido passa durante o seu ciclo de vida;
- Notificações de acompanhamento, mantendo informados todos os envolvidos com um pedido de mudança sobre alterações recebidas durante o ciclo de vida;
- Possibilitar o uso de campos personalizados, caso seja necessário acrescentar mais informações para atender a uma necessidade específica do projeto;
- Rastreamento de mudanças identificando todas as suas dependências;
- Relatórios personalizados para consulta de toda a evolução dos pedidos de mudança.

Trac

A ferramenta *Trac* é uma das mais completas. Baseada na Web é implementada como um CGI ou como um programa independente. Foi desenvolvida na linguagem de programação *Phyton*, e possui alguns diferenciais como acompanhamento da evolução do projeto, integração com a ferramenta *Subversion* e outras ferramentas de controle de versões, arquitetura de *plugins*, além de suporte para os bancos de dados *SQLite*, *PostgreSql* e *MySql*.

No endereço *www.hosted-projects.com/trac/TracDemo/Demo* podemos testar a ferramenta através de um demo. Para reportar um erro, clique no menu *New Ticket*. Na **Figura 1** vemos como a estrutura do *Trac* é bem simples, utilizando os campos:

- *Your email or username:* responsável em reportar um problema ou uma nova necessidade do usuário do software;
- *Short summary:* definição objetiva do problema encontrado ou nova necessidade;
- *Type:* tipo do que está sendo reportado, um problema ou uma nova necessidade;
- *Full description:* descrição completa do problema ou nova necessidade;
- *Priority: usado para marcar a urgência da resolução do que está sendo reportado;*
- *Component:* subsistema ou módulo de um produto;
- *Severity:* MOSTRA *o quanto é relevante o que está sendo reportado;*
- *Version:* indica a versão do software que ocorreu o problema ou a nova necessidade;
- *Assign to:* responsável pela resolução do problema ou implementação da nova necessidade;
- *Cc: lista de pessoas que podem ser informadas sobre o que está sendo reportado;*
- *Keywords:* palavras-chave para categorizar *o que está sendo reportado.*

Uma de suas características mais interessantes é a formatação *Wiki*. Um *Wiki* é utilizado para identificar um tipo específico de coleção de documentos em hipertexto ou o conteúdo de um documento que pode ficar disponível a qualquer momento através de um navegador.

Todas as informações adicionais e modificações feitas após a criação do *ticket* são mantidas, gerando um histórico da evolução do erro. Dessa forma, as mudanças que foram solicitadas e as

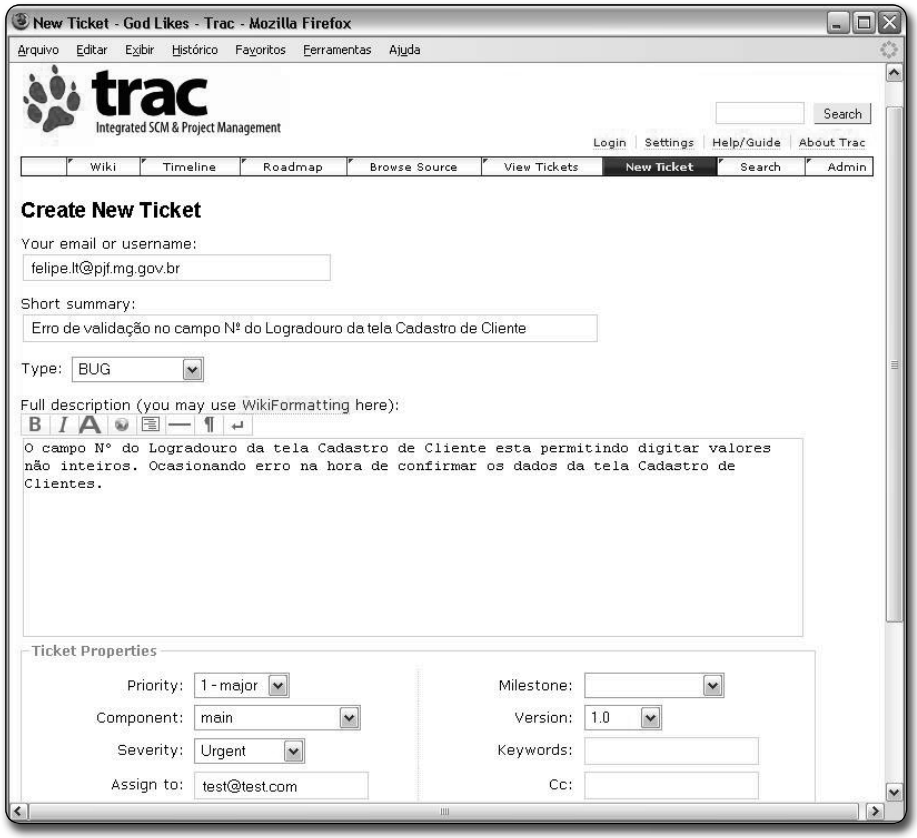


Figura 1. Visualização de um Ticket na ferramenta Trac

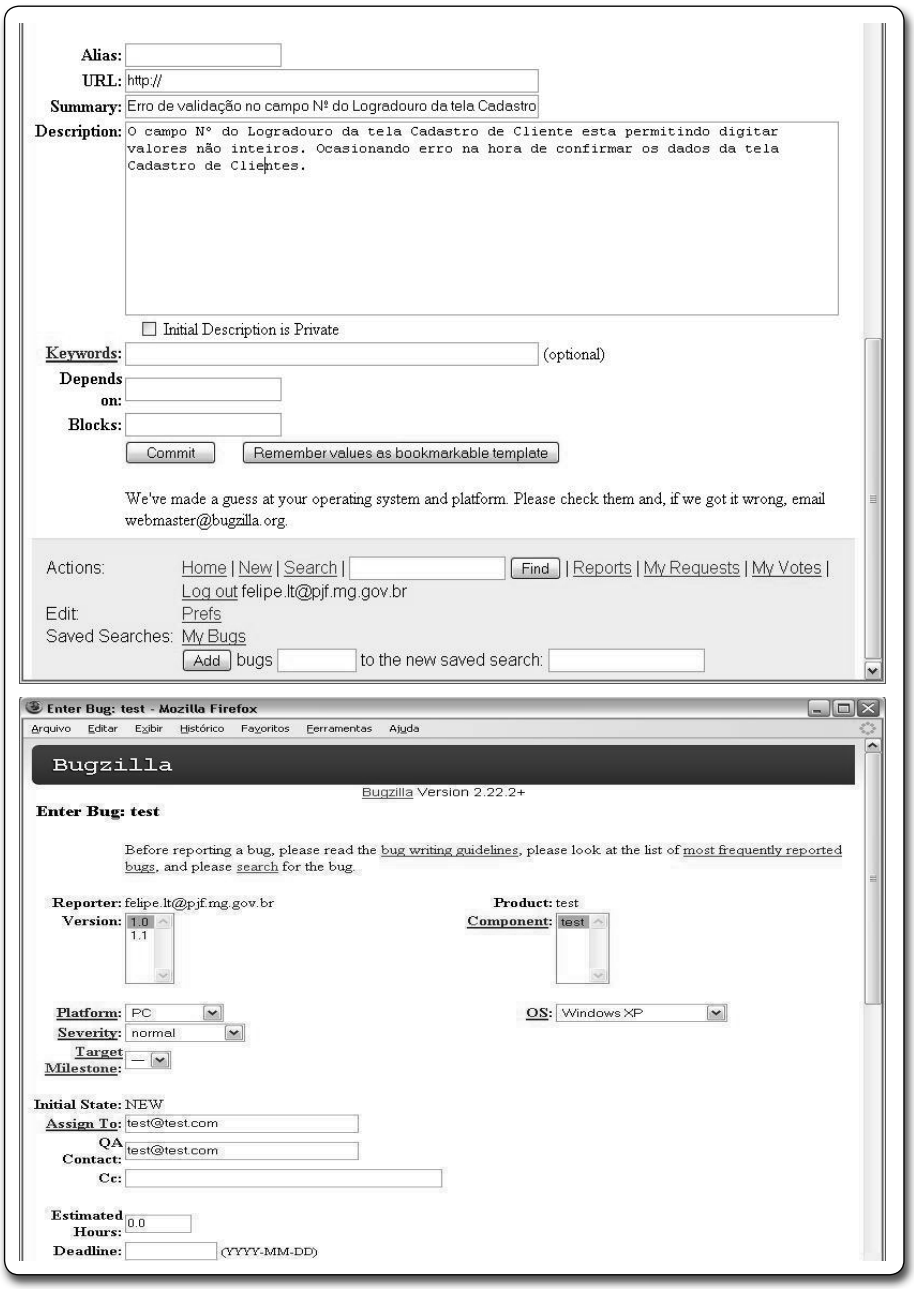


Figura 2. Visualização do Enter Bug

correções feitas no software a partir delas são rastreadas pelos *links* criados através da formatação *Wiki*.

Bugzilla

A ferramenta *Bugzilla* é uma da mais utilizadas entre os desenvolvedores, porém é de mais difícil instalação e, na instalação *default*, ainda precisa de uma grande quantidade de informações para descrever um problema.

Ela também é baseada na Web e dá suporte ao desenvolvimento do projeto *Mozilla*, permitindo o rastreamento de alterações e servindo também como plataforma para pedidos de novas solicitações.

No endereço *landfill.bugzilla.org/bugzilla-2.22-branch* podemos testar a ferramenta através de um demo. Primeiramente, para reportar um erro é necessário criar uma conta no *Bugzilla*, depois clicar em *Enter a new bug>test product*, conforme **Figura 2**.

Além de possuir alguns campos com as mesmas características da ferramenta *Trac*, também se pode destacar:

- *Reporter*: responsável em reportar um problema ou nova necessidade do usuário do software;
- *Product:* qual software o problema ou nova necessidade se refere;
- *Platform e OS*: indica o ambiente onde o problema ou nova necessidade ocorreu;
- *Initial State*: estado que um problema ou nova solicitação se encontra, por exemplo, informa se o problema é novo;
- Estimated Hours: estimativa do esforço gasto para correção do problema ou implementação da nova necessidade;

• Deadline: previsão para o concerto do problema ou implementação da nova necessidade;

• *URL*: uma URL associada ao que está sendo reportado;

• *Depends on*: informa dependências entre problemas ou novas necessidades.

Bug Tracking na prática

Para demonstrar na prática a utilização de *Bug Tracking* em um sistema, este artigo vai apresentar a criação de uma aplicação de exemplo contendo uma janela simples de Cadastro de Cliente, além de uma opção no menu da aplicação para que o usuário possa reportar erros ou novas necessidades diretamente para a equipe de desenvolvimento.

Além disso, quando um erro não tratado na aplicação ocorrer, esse também será capturado e enviado automaticamente para os desenvolvedores da aplicação. Primeiramente, temos que criar um banco de dados InterBase/Firebird com o nome de "BUGTRACKING.GDB" contendo as tabelas CLIENTE e ERRO, mostradas no *script* da **Listagem 1**, além dos *Generators* e *Stored Procedures*..

A seguir, vamos criar uma nova aplicação no Delphi, salvar a unit com o nome de "ufrmPadrao.pas" e o projeto como "BugTracking.dpr". Essa unit será definida como o formulário padrão da aplicação, e os novos formulários serão criados a partir dela, facilitando o processo de desenvolvimento.

Para isso, adicione o formulário ao repositório do Delphi, clicando com o botão direito e escolhendo o item *Add to Repository* configurando-o como apresentado na **Figura 3**.

Dando continuidade ao projeto, para acessar o banco de dados criado, devemos criar um Data Module em *File>New>Data Module* e salvar com o nome de "udtmdlPrincipal.pas". Adicione os componentes *SQLConnection*, *SQLStoredProc*, *SQLDataSet*, *DataSetProvider* e *ClientDataSet* (**Figura 4**). Devemos então definir seus nomes e propriedades conforme apresentado na **Tabela 1**, respectivamente.

Capturando erros da aplicação

Com o Data Module salvo ao projeto, podemos implementar a função "CapturaErro" na seção *private* em *ufrmPa-*

Listagem 1. Objetos do banco de dados

```
CREATE TABLE CLIENTE (
    ID_CLIENTE   INTEGER NOT NULL,
    NOME         VARCHAR(70),
    LOGRADOURO   VARCHAR(50),
    NUMERO       INTEGER,
    COMPLEMENTO  VARCHAR(10),
    BAIRRO       VARCHAR(30));
ALTER TABLE CLIENTE ADD CONSTRAINT PK_CLIENTE
PRIMARY KEY (ID_CLIENTE);
CREATE TABLE ERRO (
    ID_ERRO     INTEGER NOT NULL,
    SUMARIO     VARCHAR(50),
    DESCRICAO   BLOB SUB_TYPE 0 SEGMENT SIZE 500,
    USUARIO     VARCHAR(30),
    DATA        TIMESTAMP,
    TIPO        INTEGER,
    CONTROLE    VARCHAR(30),
    FORMULARIO  VARCHAR(30),
    SEVERIDADE  INTEGER,
    STATUS      INTEGER,
    VERSAO      VARCHAR(10));
ALTER TABLE ERRO ADD CONSTRAINT PK_ERRO
PRIMARY KEY (ID_ERRO);
CREATE GENERATOR GEN_CLIENTE_ID;
CREATE GENERATOR GEN_ERRO_ID;
CREATE PROCEDURE SP_GEN_CLIENTE_ID
RETURNS (
    ID INTEGER)
AS
BEGIN
  ID = GEN_ID(GEN_CLIENTE_ID, 1);
  SUSPEND;
END
CREATE PROCEDURE SP_GEN_ERRO_ID
RETURNS (
    ID INTEGER)
AS
BEGIN
  ID = GEN_ID(GEN_ERRO_ID, 1);
  SUSPEND;
END
```

Listagem 2. Função para Capturar erros não tratados pelo sistema

```
uses udtmdlPrincipal;
...
procedure TfrmPadrao.CapturarErro(Sender: TObject;
  Excptn: Exception);
begin
  dtmdlPrincipal.clntdtstPrincipal.Close;
  dtmdlPrincipal.sqldtstPrincipal.CommandText := 'select * from ERRO';
  dtmdlPrincipal.clntdtstPrincipal.Open;
  dtmdlPrincipal.clntdtstPrincipal.Insert;
  dtmdlPrincipal.sqlspPrincipal.StoredProcName := 'SP_GEN_ERRO_ID';
  dtmdlPrincipal.sqlspPrincipal.ExecProc;
  dtmdlPrincipal.clntdtstPrincipal.FieldByName(
    'ID_ERRO').AsInteger := dtmdlPrincipal.sqlspPrincipal.Params[0].AsInteger;
  dtmdlPrincipal.clntdtstPrincipal.FieldByName('USUARIO').AsString := 'admin';
  dtmdlPrincipal.clntdtstPrincipal.FieldByName('DATA').AsDateTime  := Now;
  dtmdlPrincipal.clntdtstPrincipal.FieldByName('TIPO').AsInteger := 0;
  dtmdlPrincipal.clntdtstPrincipal.FieldByName(
    'SUMARIO').AsString := 'Erro do Sistema em ' + Screen.ActiveForm.Caption;
  dtmdlPrincipal.clntdtstPrincipal.FieldByName('DESCRICAO').AsString := Excptn.message;
  dtmdlPrincipal.clntdtstPrincipal.FieldByName('FORMULARIO').AsString := Screen.ActiveForm.Name;
  dtmdlPrincipal.clntdtstPrincipal.FieldByName(
    'CONTROLE').AsString := Screen.ActiveControl.Name;
  dtmdlPrincipal.clntdtstPrincipal.FieldByName('SEVERIDADE').AsInteger := 0;
  dtmdlPrincipal.clntdtstPrincipal.FieldByName('STATUS').AsInteger:= 0;
  dtmdlPrincipal.clntdtstPrincipal.FieldByName('VERSAO').AsString := 'v1.0';
  dtmdlPrincipal.clntdtstPrincipal.ApplyUpdates(0);
  MessageDlg(Excptn.message+'. Este erro será gravado e poderá ser enviado '+
    'posteriormente para a equipe de Desenvolvimento.', mtError, [mbOK], 0);
end;
```

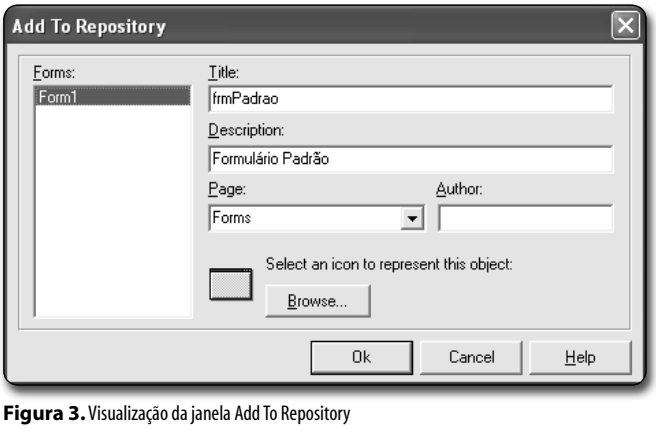


Figura 3. Visualização da janela Add To Repository

Nome	Propriedade	Valor
sqlcnctnConexao	Params	Configurar o caminho do banco
sqlspPrincipal	SQLConnection	sqlcnctnConexao
sqldtstPrincipal	SQLConnection	sqlcnctnConexao
dtstprvdrPrincipal	DataSet	sqldtstPrincipal
clntdtstPrincipal	ProviderName	dtstprvdrPrincipal

Tabela 1. Configurações dos componentes de acesso ao banco de dados

Campo	Valor	Corresponde
ID_ERRO	sqlspPrincipal.Params[0].AsInteger	Executar *procedure* auto-incremento
USUARIO	admin	Valor *default* do usuário do sistema
DATA	Now	Momento que ocorreu o erro
TIPO	0	0=Erro / 1=Nova Solicitação
SUMARIO	Screen.ActiveForm.Caption	Nome da janela que ocorreu o erro
DESCRICAO	Exception.message	Mensagem de erro que foi exibida
FORMULARIO	Screen.ActiveForm.Name	Formulário que ocorreu o erro
CONTROLE	Screen.ActiveControl.Name	Componente que gerou o erro
SEVERIDADE	0	0=Alta / 1=Média / 2=Baixa
STATUS	0	0=Em Aberto / 1=Corrigido / 3=Rejeitado
VERSAO	V1.0	Valor *default* da versão do software

Tabela 2. Valores manipulados pela função CapturarErro

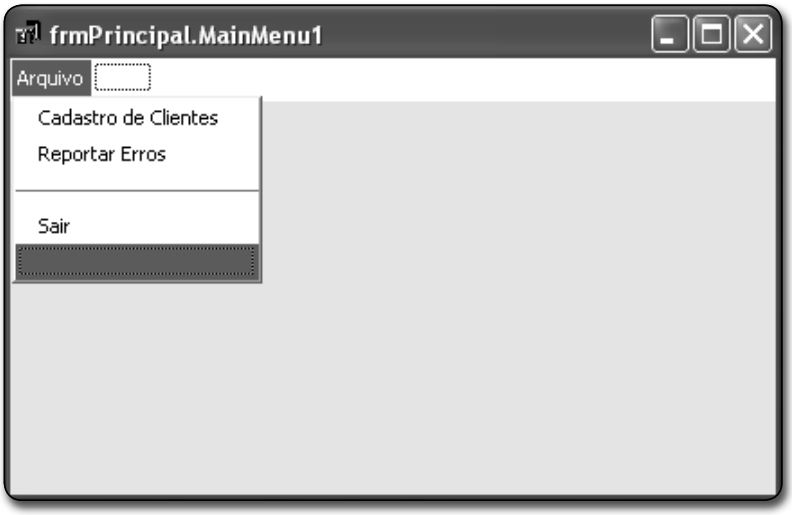


Figura 5. Visualização do menu da janela principal da aplicação

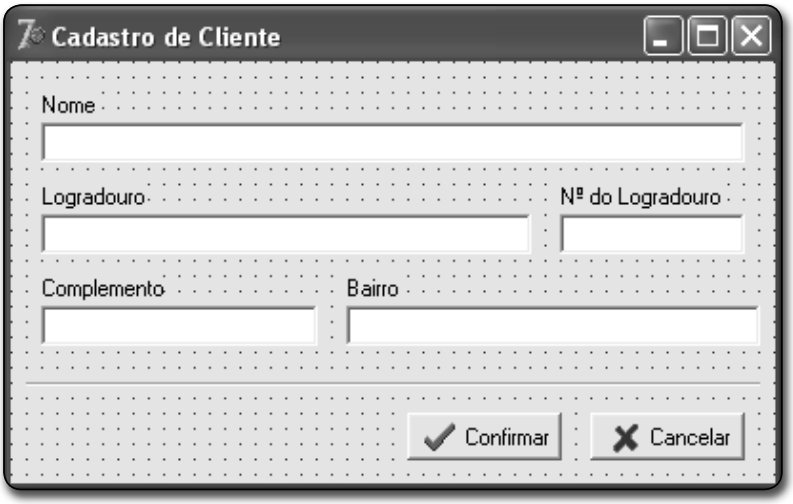


Figura 6. Visualização da tela Cadastro de Cliente

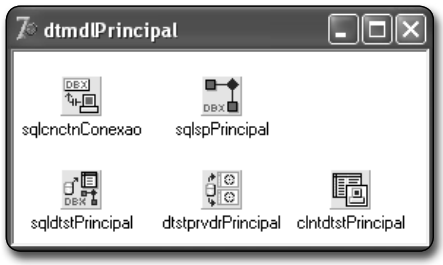


Figura 4. Visualização do Data Module

drao criado anteriormente, conforme a **Listagem 2**.

Essa função capturará o máximo de informações possíveis geradas por um erro inesperado no sistema e gravará na tabela ERRO. Dessa forma, um erro ocorrido poderá ser enviado posteriormente para a equipe de desenvolvimento do software.

Na **Tabela 2** são apresentadas algumas características importantes da *CapturaErro*.

Para finalizar a codificação desse formulário, adicione no *OnCreate* do mesmo, o seguinte código:

```
Application.OnException := CapturarErro;
```

Com o formulário padrão adicionado ao repositório do Delphi, podemos criar uma tela principal para o sistema, em *File>New>Other*. Selecione a aba *Form*, escolha *frmPadrao* e marque o item *Inherit*, para que o novo formulário herde todas as características do formulário padrão, como a *CapturaErro*. Adicione um *MainMenu* com os itens da **Figura 5** e salve a unit com o nome de "ufrmPrincipal.pas".

Também devemos criar uma tela simples de cadastro, seguindo os mesmos passos da tela principal, salvando a unit como "ufrmCadastroCliente.pas" e configurar um *layout* semelhante a **Figura 6**.

Na *ufrmCadastroCliente* no evento *OnClick* do *Confirmar*, digite o código apresentado na **Listagem 3**, para gravação dos dados (atente para o nome dos componentes de tela). Neste exemplo, veja que não foi implementado nenhum tratamento dos valores informados no campo Nº do Logradouro.

Em nosso banco de dados o campo NUMERO é do tipo INTEGER, podendo ocorrer de um usuário digitar um valor não numérico nesse campo, gerando erro na hora de gravar os dados.

Listando os erros capturados

Para visualizar os erros cadastrados devemos criar um novo formulário usando a mesma técnica anterior, e salvar a *unit* como "ufrmListaErros.pas". Então adicione os componentes *SQLStoredProc*, *SQLDataSet*, *DataSetProvider*, *ClientDataSet*, *DataSouce* e um *DBGrid* para exibir a lista com os erros e novas solicitações gerados no sistema, conforme a **Figura 7**.

A configuração dos componentes é parecida com a do Data Module mas com algumas diferenças, começando pelos nomes que terão os finais "Grade" como, por exemplo, o nome do *DataSouce* será *dtsrcGrade*.

Também devemos deixar a propriedade *SQLConnection* do *sqldtstGrade* vazia, pois o valor será atribuído em tempo de execução pelo código da **Listagem 4**, através do evento *OnCreate* do formulário, fazendo a conexão com o banco de dados.

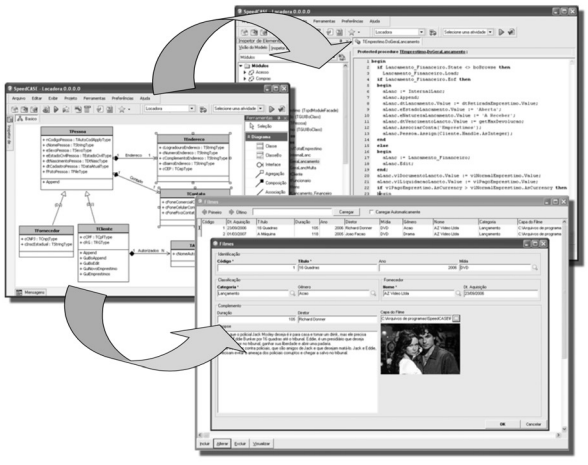
Listagem 3. Evento OnClick do Confirmar da janela Cadastro de Cliente

```
uses udtmdlPrincipal;
...
procedure TfrmCadastroCliente.btnConfirmarClick(Sender: TObject);
begin
  inherited;
  dtmdlPrincipal.clntdtstPrincipal.Close;
  dtmdlPrincipal.sqldtstPrincipal.CommandText := 'select * from CLIENTE';
  dtmdlPrincipal.clntdtstPrincipal.Open;
  dtmdlPrincipal.clntdtstPrincipal.Insert;
  dtmdlPrincipal.sqlspPrincipal.StoredProcName := 'SP_GEN_CLIENTE_ID';
  dtmdlPrincipal.sqlspPrincipal.ExecProc;
  dtmdlPrincipal.clntdtstPrincipal.FieldByName(
    'ID_CLIENTE').AsInteger := dtmdlPrincipal.sqlspPrincipal.Params[0].AsInteger;
  dtmdlPrincipal.clntdtstPrincipal.FieldByName('NOME').AsString := edtNome.Text;
  dtmdlPrincipal.clntdtstPrincipal.FieldByName('LOGRADOURO').AsString := edtLogradouro.Text;
  dtmdlPrincipal.clntdtstPrincipal.FieldByName('NUMERO').AsString := edtNumero.Text;
  dtmdlPrincipal.clntdtstPrincipal.FieldByName('COMPLEMENTO').AsString := edtComplemento.Text;
  dtmdlPrincipal.clntdtstPrincipal.FieldByName('BAIRRO').AsString := edtBairro.Text;
  dtmdlPrincipal.clntdtstPrincipal.ApplyUpdates(0);
  Application.MessageBox('Os dados foram incluídos.', 'Bug Report', MB_IconInformation);
end;
```

Listagem 4. Código do OnCreate do formulário de exibição de erros

```
uses udtmdlPrincipal;
...
dtmdlPrincipal := TdtmdlPrincipal(
  application.FindComponent('dtmdlPrincipal') as TDataModule);
sqldtstGrade.SQLConnection := dtmdlPrincipal.sqlcnctnConexao;
clntdtstGrade.Close;
sqldtstGrade.CommandText := 'select * from ERRO';
clntdtstGrade.Open;
```


Reduza o tempo de desenvolvimento em até 80%



Speed CASE™

Ambiente de Análise e Implementação Integrados.
Interface Gráfica e Banco de Dados gerados a partir do modelo.

Ferramenta de produtividade para desenvolvimento de software

Conheça a versão Free!

Persistência com:
Interbase
Firebird
MySQL
MS SQL Server

Compatível com:
Turbo Delphi
Delphi
Kylix

Desenvolva para:
Windows
Linux

Faça do tempo seu diferencial

TecnoSpeed TI - Fone: (44) 3028-3749 - **www.speedcase.com.br**

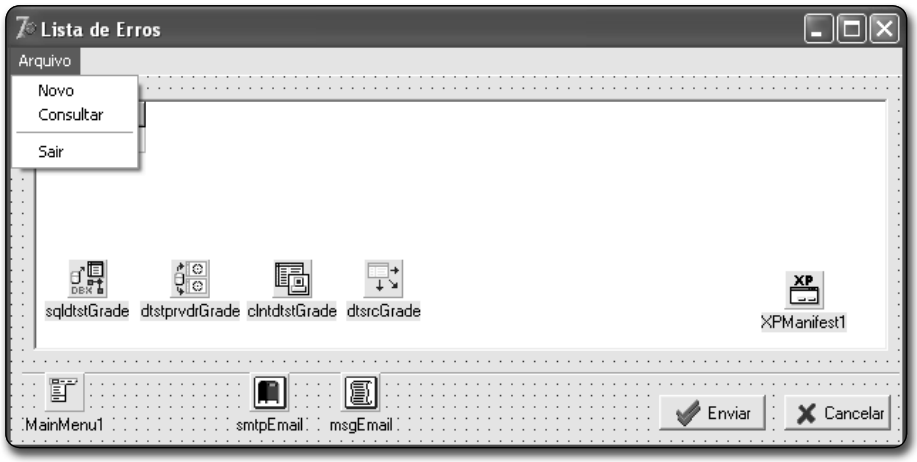


Figura 7. Visualização da janela Lista de Erros

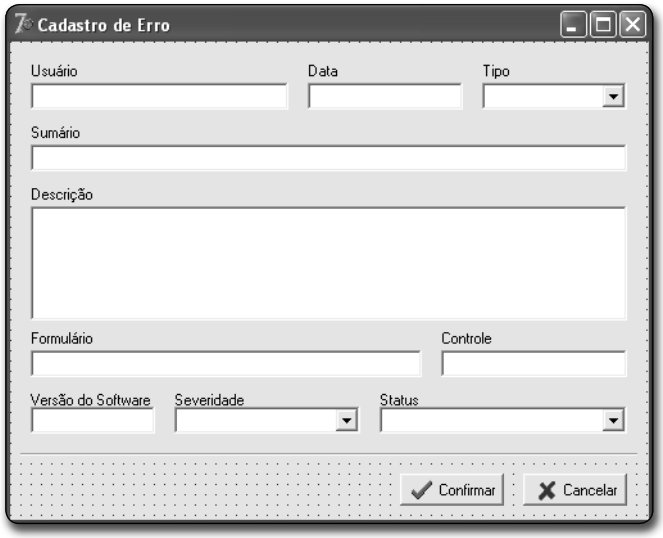


Figura 8. Visualização da janela Cadastro de Erro

Listagem 5. Evento OnClick do Confirmar, Tela Cadastro de Erro

```
uses udtmdlPrincipal;
...
procedure TfrmCadastroErro.btnConfirmarClick(Sender: TObject);
begin
  inherited;
  dtmdlPrincipal.clntdtstPrincipal.Close;
  dtmdlPrincipal.sqldtstPrincipal.CommandText := 'select * from ERRO';
  dtmdlPrincipal.clntdtstPrincipal.Open;
  dtmdlPrincipal.clntdtstPrincipal.Insert;
  dtmdlPrincipal.sqlspPrincipal.StoredProcName := 'SP_GEN_ERRO_ID';
  dtmdlPrincipal.sqlspPrincipal.ExecProc;
  dtmdlPrincipal.clntdtstPrincipal.FieldByName(
    'ID_ERRO').AsInteger := dtmdlPrincipal.sqlspPrincipal.Params[0].AsInteger;
  dtmdlPrincipal.clntdtstPrincipal.FieldByName('USUARIO').AsString := edtUsuario.Text;
  dtmdlPrincipal.clntdtstPrincipal.FieldByName('DATA').AsString := edtData.Text;
  if cmbxTipo.ItemIndex <> -1 then
  begin
    dtmdlPrincipal.clntdtstPrincipal.FieldByName('TIPO').AsInteger := cmbxTipo.ItemIndex;
  end;
  dtmdlPrincipal.clntdtstPrincipal.FieldByName('SUMARIO').AsString := edtSumario.Text;
  dtmdlPrincipal.clntdtstPrincipal.FieldByName('DESCRICAO').AsString := mDescricao.Lines.Text;
  dtmdlPrincipal.clntdtstPrincipal.FieldByName('FORMULARIO').AsString := edtFormulario.Text;
  dtmdlPrincipal.clntdtstPrincipal.FieldByName('CONTROLE').AsString := edtControle.Text;
  if cmbxSeveridade.ItemIndex <> -1 then
  begin
    dtmdlPrincipal.clntdtstPrincipal.FieldByName('SEVERIDADE').AsInteger := cmbxSeveridade.ItemIndex;
  end;
  if cmbxStatus.ItemIndex <> -1 then
  begin
    dtmdlPrincipal.clntdtstPrincipal.FieldByName('STATUS').AsInteger:= cmbxStatus.ItemIndex;
  end;
  dtmdlPrincipal.clntdtstPrincipal.FieldByName('VERSAO').AsString := edtVersao.Text;
  dtmdlPrincipal.clntdtstPrincipal.ApplyUpdates(0);
  Application.MessageBox('Os dados foram incluídos.', 'Bug Tracking', MB_IconInformation);
end;
```

No código anterior, também indicamos o valor do *CommandText* do *sqldtstGrade*. Faça a configuração normal dos componentes para exibir os dados no *DBGrid*. Para o envio automático da lista com os erros para a equipe de desenvolvimento, adicionamos no formulário os componentes *idSMTP* da paleta *IndyClients* e *idMessage* da paleta *Indy Misc* (o código será visto adiante no artigo).

Cadastrando erros

Para reportar manualmente erros ou até mesmo solicitar melhorias no sistema, um novo formulário deverá ser construído, repetindo os mesmos passos da criação do formulário anterior, salvando a *unit* como "ufrmCadastroErro.pas".

Esse formulário foi criado seguindo algumas características da ferramenta *Trac* (**Figura 8)**.

Ele também será usado para exibir os dados de um erro já cadastrado no banco. O evento *OnClick* do *Confirmar* deve ser implementado conforme a **Listagem 5**, que é bem parecido com a *CapturarErro* mas, ao contrário dessa função que registra automaticamente os problemas ocorridos, esse evento gravará os dados informados manualmente pelo cliente.

No *OnShow* vamos verificar se queremos cadastrar ou exibir um erro, utilizando o código da **Listagem 6.**

As variáveis *pCodErro* e *pOperacao* são do tipo *string* e devem ser declaradas na seção *private*. Por fim, implemente o método *Parâmetro* no formulário, conforme o código da **Listagem 7**.

Agora, com as janelas do sistema prontas, devemos programar no evento *OnClick* do *MenuItem* em *ufrmPrincipal* o código da **Listagem 8**, para exibir o Cadastro de Cliente e fazer o mesmo para exibir a janela de Lista de Erros, mas é necessário alterar o nome do formulário para "frmListaErros".

Fazer o mesmo na *ufrmListaErros* para exibir a tela de Cadastro de Erros, alterando o nome do formulário para "frmCadastroErro".

Para o formulário de listagem de erros, para chamar o mesmo, para adicionar um novo erro, devemos utilizar o código da **Listagem 9**.

Se quisermos visualizar as informações do erro, no item de menu *Consultar*, use o

código da **Listagem 10**.

Antes de executar a aplicação, vá na opção *Project>Options*, na aba *Form* do *Project Options for BugTracking.exe* deixe na lista *Auto-create forms* somente o *dtmdlPrincipal* e o *frmPrincipal*. Compile e execute o projeto.

No menu da janela principal, abra o Cadastro de Cliente e preencha todos os campos mas, no campo Nº do Logradouro digite um valor não inteiro, e confirme os dados. No momento que clicar no *Confirmar*, o código implementado no evento *OnClick* tentará gravar um valor não inteiro no campo do tipo INTEGER no banco de dados, ocorrendo um erro não tratado pelo sistema (**Figura 9**). Assim o *CapturarErro* será chamado, gravando o erro automaticamente no banco de dados.

Para visualizar o erro, clique na opção *Reportar Erro* no menu da janela principal do sistema. Os erros existentes serão exibidos em uma lista, podendo também visualizar os detalhes de um erro, selecionando-o na lista e acessando o menu *Consultar*, da janela Lista de Erros (**Figura 10**).

Se o usuário necessitar reportar uma nova necessidade para o sistema, como por exemplo, adicionar o campo CPF na tela de Cadastro de Cliente, também será possível através da tela de Cadastro de Erro, bastando selecionar no campo Tipo o valor *Nova Solicitação* e preencher os campos com as características de sua

Listagem 6. Exibindo ou cadastrando um erro

```
procedure TfrmCadastroErro.FormShow(Sender: TObject);
begin
  inherited;
  if pOperacao = 'Novo' then
  begin
    edtData.Text := DateTimeToStr(now);
    edtUsuario.Text := 'admin';
    edtVersao.Text := 'v1.0';
  end
  else
  begin
    pnlConsulta.Enabled := False;
    btnConfirmar.Visible := False;
    dtmdlPrincipal.clntdtstPrincipal.Close;
    dtmdlPrincipal.sqldtstPrincipal.CommandText := 'select * from ERRO where ID_ERRO = '+ pCodErro;
    dtmdlPrincipal.clntdtstPrincipal.Open;
    edtUsuario.Text := dtmdlPrincipal.clntdtstPrincipal.FieldByName('USUARIO').AsString;
    edtData.Text := dtmdlPrincipal.clntdtstPrincipal.FieldByName('DATA').AsString;
    if dtmdlPrincipal.clntdtstPrincipal.FieldByName('TIPO').AsInteger <> -1 then
    begin
      cmbxTipo.ItemIndex := dtmdlPrincipal.clntdtstPrincipal.FieldByName('TIPO').AsInteger;
    end;
    edtSumario.Text := dtmdlPrincipal.clntdtstPrincipal.FieldByName('SUMARIO').AsString;
    Descricao.Lines.Add(dtmdlPrincipal.clntdtstPrincipal.FieldByName('DESCRICAO').AsString);
    edtFormulario.Text := dtmdlPrincipal.clntdtstPrincipal.FieldByName('FORMULARIO').AsString;
    edtControle.Text := dtmdlPrincipal.clntdtstPrincipal.FieldByName('CONTROLE').AsString;
    edtVersao.Text := dtmdlPrincipal.clntdtstPrincipal.FieldByName('VERSAO').AsString;
    if dtmdlPrincipal.clntdtstPrincipal.FieldByName('STATUS').AsInteger <> -1 then
    begin
      cmbxStatus.ItemIndex := dtmdlPrincipal.clntdtstPrincipal.FieldByName('STATUS').AsInteger;
    end;
    if dtmdlPrincipal.clntdtstPrincipal.FieldByName('SEVERIDADE').AsInteger <> -1 then
    begin
      cmbxSeveridade.ItemIndex :=
        dtmdlPrincipal.clntdtstPrincipal.FieldByName('SEVERIDADE').AsInteger;
    end;
  end;
end;
```

Listagem 7. Configurando o parâmetro para exibir ou cadastrar um erro

```
procedure TfrmCadastroErro.Parametro(CodErro, Operacao: string);
begin
  pCodErro := CodErro;
  pOperacao := Operacao;
end;
```

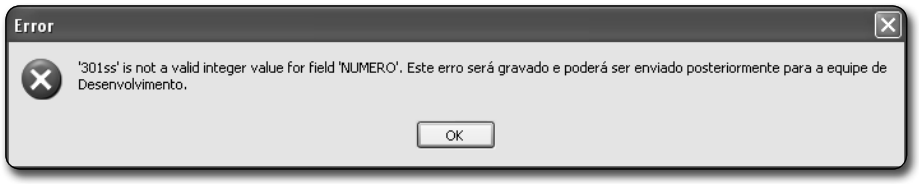


Figura 9. Visualização da mensagem do erro gerada

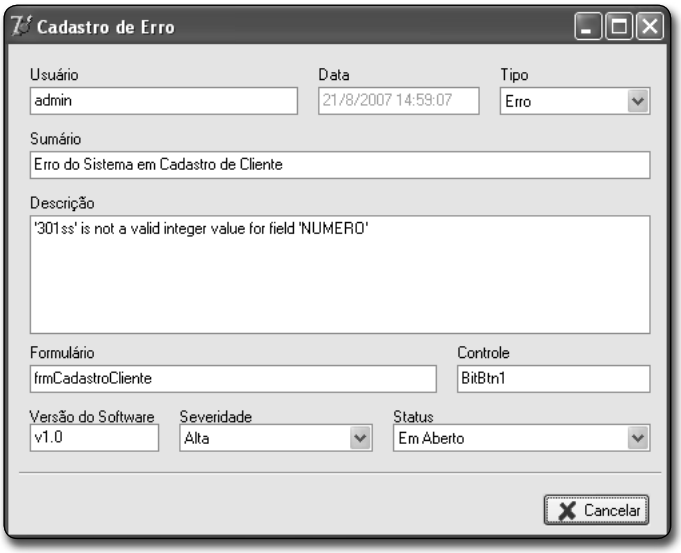


Figura 10. Visualização de um erro

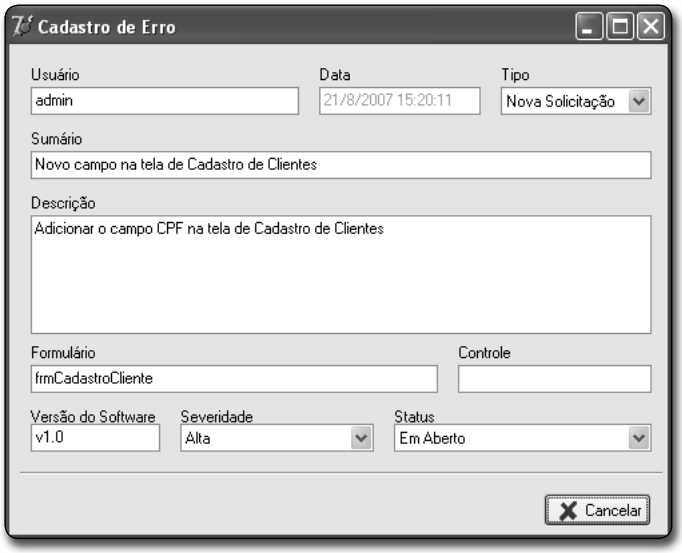


Figura 11. Visualização dos erros gerados

Listagem 8. Evento OnClick do MenuItem

```
uses ufrmCadastroCliente, ufrmListaErros;
...
procedure TfrmPrincipal.CadastrodeCliente1Click(Sender: TObject);
begin
  inherited;
  try
    frmCadastroCliente := TfrmCadastroCliente.Create(Self);
    frmCadastroCliente.ShowModal;
  finally
    frmCadastroCliente.Release;
  end;
end;
procedure TfrmPrincipal.ReportarErros1Click(Sender: TObject);
begin
  inherited;
  try
    frmListaErros := TfrmListaErros.Create(Self);
    frmListaErros.ShowModal;
  finally
    frmListaErros.Release;
  end;
end;
```

Listagem 9. Chamando o formulário para cadastro de erro

```
try
  frmCadastroErro := TfrmCadastroErro.Create(Self);
  frmCadastroErro.Parametro('0','Novo');
  frmCadastroErro.ShowModal;
finally
  frmCadastroErro.Release;
end;
```

Listagem 10. Consultando um erro cadastrado

```
if clntdtstGrade.FieldByName('ID_ERRO').IsNull then
begin
  Application.MessageBox(
    'É necessário selecionar um erro na lista.', 'Bug Report', MB_ICONWARNING);
end
else
begin
  try
    frmCadastroErro := TfrmCadastroErro.Create(Self);
    frmCadastroErro.Parametro(clntdtstGrade.FieldByName('ID_ERRO').AsString,'Consultar');
    frmCadastroErro.ShowModal;
  finally
    frmCadastroErro.Release;
  end;
end;
```

Listagem 11. Enviar e-mail com os dados dos erros

```
procedure TfrmListaErros.btnEnviarClick(Sender: TObject);
var
  bug:TextFile;
  i: Integer;
begin
  try
    clntdtstGrade.First;
    AssignFile(bug, ExtractFilePath(Application.ExeName)+'\ListaErros.bug');
    Rewrite(bug);
    while not clntdtstGrade.Eof do
    begin
      if clntdtstGrade.FieldByName('STATUS').AsInteger = 0 then
      begin
        for i:= 0 to clntdtstGrade.Fields.Count-1 do
        begin
          Write(bug, clntdtstGrade.Fields[i].Value);
          Write(bug, ',');
        end;
      end;
      clntdtstGrade.Next;
    end;
    Closefile(bug);
    smtpEmail.Host := 'smtp... ';
    smtpEmail.Port := 25;
    smtpEmail.Username := 'login de email do usuário';
    smtpEmail.Password := 'senha do usuário';
    smtpEmail.AuthenticationType := atLogin;
    msgEmail.From.Text := 'email do usuário';
    msgEmail.Recipients.EMailAddresses := 'email da equipe de desenvolvimento';
    msgEmail.Priority := mpHigh;
    msgEmail.Subject := 'Lista de Erros';
    msgEmail.Body.Text := 'Segue em anexo erros do sistema Bug Tracking.';
    TIdAttachment.create(msgEmail.MessageParts,
      TFileName(ExtractFilePath(Application.ExeName)+'\ListaErros.bug'));
    smtpEmail.Connect;
    try
      smtpEmail.Send(msgEmail);
    finally
      smtpEmail.Disconnect;
    end;
    Application.MessageBox('Email enviado com sucesso!', 'Bug Tracking',MB_ICONINFORMATION + MB_OK);
  except
    Application.MessageBox('Ocorreu um problema ao enviar o email.',
      'Bug Tracking',MB_ICONERROR + MB_OK);
  end;
end;
```

nova necessidade.

Todos os erros, gerados automaticamente ou informados manualmente, bem como as novas solicitações cadastradas são exibidas na lista de erros, conforme **Figura 11**.

Para enviar tudo o que foi reportado pelo sistema para a equipe de desenvolvimento por e-mail, basta implementar no evento *OnClick* do *Enviar* do *frmListaErros* o código apresentado na **Listagem 11**.

O código gerará um arquivo com a extensão BUG com as informações dos erros e novas solicitações geradas no sistema, mas somente serão gravados no arquivo os registros que possuem Status *Em Aberto*, permitindo ao usuário o acompanhamento do que foi reportado.

O arquivo *ListaErros.bug* será gravado no mesmo local de onde o sistema está sendo executado, e enviará o mesmo automaticamente para a equipe de desenvolvimento do software por e-mail.

> **Nota:** Configure as contas de e-mail no código do evento *OnClick* do *Confirmar*. Dependendo da conta de e-mail utilizada será necessário atualizar o Indy para uma versão mais atual.

Conclusão

A comunicação é um fator que não pode ser desconsiderado ao longo de todo o ciclo de vida de um software. Mesmo depois da entrega do produto ao usuário final é desejável permitir interação com o projeto.

O uso de ferramentas para o rastreamento e o controle das mudanças ocorridas no projeto é um passo importante, gerando melhoria na qualidade tanto no processo de desenvolvimento do software quanto no produto gerado. ●

Unimos a inovação e excelência da **Alog** com
o pioneirismo da **.comDomínio** para criar a
Alog Data Centers do Brasil:

LÍDER EM HOSTING GERENCIADO®

ALOG data centers + .comDominio DATA CENTERS

CONHEÇA OS NÚMEROS DESTA LIDERANÇA

2 Data Centers de Classe Mundial
200 Profissionais Dedicados
800 Clientes Corporativos
3.000 Servidores Hospedados
8.000 m² de Área Construída

www.alog.com.br | 0800 282 3330

ALOG DATA CENTERS DO BRASIL

Hosting Gerenciado® | Colocation | Contingência | Email Corporativo | Conectividade | Serviços Profissionais

Contas a Pagar e Cobrança

Crie um sistema completo com Delphi, Firebird 2.0 e dbExpress - Parte 2

Adriano Santos
(artes@doiscliques.com)
é desenvolvedor Delphi desde 1998. Professor e programador PHP. Bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É colunista e membro da Comissão Editorial da revista ClubeDelphi. Mantém o blog Delphi to Delphi (www.delphitodelphi.blogspot.com) com dicas, informações e tudo sobre desenvolvimento Delphi.

Na edição anterior de ClubeDelphi, iniciamos o curso de Contas a Pagar e Cobrança desenvolvendo as primeiras telas que farão parte do sistema chamado *SysPague*, onde vimos as principais dicas e detalhes referentes ao projeto.

Neste artigo, vamos criar mais uma tela de cadastro, que sem dúvida é uma das principais portas de entrada de dados, o de Contas a Pagar. Nela, o usuário final poderá incluir, alterar e baixar os lançamentos de contas a pagar, tais como contas de água, luz, telefone, gastos com office-boy e moto-boy etc.

Para iniciarmos nosso artigo, veremos rapidamente o que foi criado no exemplo anterior:

- Banco de dados Firebird 2.0;
- Tela principal;
- Cadastro de Clientes;
- Cadastro de Fornecedores;
- Cadastro de Contas Correntes;
- Data Module principal;

Daremos continuidade ao curso desenvolvendo o cadastro de contas a pagar como mencionado anteriormente, para isso, precisamos alterar o banco de dados, incluindo mais uma tabela.

Alterando o banco de dados de exemplo

Inicie o *IBExpert* e vamos criar a tabela *Contas_Pagar*. Após conectar ao banco de dados no *IBExpert*, clique com o botão direito sobre o item *Tables* e escolha a opção *New Table*. No alto da janela, aparecerá o nome da tabela sugerido, troque-o para "Contas_Pagar".

Em seguida para criar um campo, digite o nome em *Field Name*, selecione seu tipo em *Field Type* e marque o item *Not Null* (caso seja necessário). Caso prefira, execute o *script* da **Listagem 1**.

Nota: O campo CODIGO é do tipo *VarChar* para armazenar o número da conta a pagar. No caso, pode ser o número de uma nota fiscal, número de referência da conta de água etc. Caso deseje, crie um campo CODIGO como autoincremento da tabela e crie um campo NUMERO_DOCUMENTO, por exemplo.

Listagem 1. Criação da tabela e índices Contas_Pagar

```
CREATE TABLE CONTAS_PAGAR (
    CODIGO        VARCHAR(20) NOT NULL,
    CNPJ          VARCHAR(18) NOT NULL,
    DESCRICAO     VARCHAR(60) NOT NULL,
    VLR_REAL      NUMERIC(15,2),
    VLR_PAGO      NUMERIC(15,2),
    JUROS         NUMERIC(15,2),
    MORA          NUMERIC(15,2),
    DT_CADASTRO   TIMESTAMP,
    DT_VECTO      TIMESTAMP,
    DT_PAGTO      TIMESTAMP,
    BANCO         INTEGER,
    AGENCIA       VARCHAR(10),
    CONTA         VARCHAR(10),
    STATUS        VARCHAR(1),
    DT_ALTERACAO  TIMESTAMP);
ALTER TABLE CONTAS_PAGAR
ADD CONSTRAINT PK_CONTAS_PAGAR
PRIMARY KEY (CODIGO, CNPJ);

ALTER TABLE CONTAS_PAGAR
ADD CONSTRAINT FK_CONTAS_PAGAR_BANCO
FOREIGN KEY (BANCO, AGENCIA, CONTA)
REFERENCES CONTAS (BANCO, AGENCIA, CONTA)
ON UPDATE CASCADE;

ALTER TABLE CONTAS_PAGAR
ADD CONSTRAINT FK_CONTAS_PAGAR_CNPJ
FOREIGN KEY (CNPJ) REFERENCES FORNECEDORES (CNPJ)
ON UPDATE CASCADE;
```

O interessante e mais importante nesse momento é que criaremos chaves estrangeiras para um maior controle de integridade dos dados manipulados pelo sistema. Note que na nova tabela temos os campos *CNPJ*, *Banco*, *Agencia* e *Conta*. O primeiro campo deverá conter o mesmo conteúdo do campo de mesmo nome, da tabela *Fornecedores*, isso porque obrigaremos o usuário final a escolher um fornecedor válido para inclusão do lançamento de contas a pagar.

Também nesses mesmos moldes, será necessário que o usuário informe o banco, agência e conta que o lançamento foi pago no ato da baixa. Feito isso, teremos que atentar ao seguinte detalhe: e se houver a necessidade de alterar o CNPJ do fornecedor, embora chave primária, ou mesmo alterar o número da conta corrente? Como ficarão nossos lançamentos no contas a pagar?

Pensando nisso usaremos um recurso bastante útil no banco de dados: *Foreing Key*, ou, *Chave Estrangeira*. As chaves estrangeiras informam para o banco, que determinado campo está associado a outro campo de outra tabela. Sendo assim, toda vez que um registro origem é alterado ou apagado, os mesmo são refletidos nas tabelas que contém essas chaves estrangeiras, mantendo a integridade dos dados.

A criação de chaves estrangeiras no banco é muito simples. Após criada a tabela *Contas_Pagar*, clique duas vezes nela e note que ao abrir, o *IBExpert* mostra algumas abas. Uma delas é a *Constraints*. Primeiramente criaremos uma chave primária para a base de dados usando a aba *Primary Key* (siga os passos a seguir, se não usou o código da **Listagem 1**).

Clique com o botão direito na área em branco e selecione *New primary key*. Em *Contraint Name* digite "PK_CONTAS_PAGAR". Clique em *On Field* e na tela que se abre, selecione os campos *Codigo* e *CNPJ* movendo-os para a direita e depois clicando no "**X**" no canto esquerdo inferior da caixa de diálogo. Repita o nome do índice em *Index Name* e marque como *Ascending* em *Index Sorting*. Por fim clique no botão com o símbolo de um raio para confirmar a criação do índice primário.

Agora criaremos nossas chaves estrangeiras. Para isso, clique na aba *Foreing Key* e em seguida clique com o botão direito em uma área em branco. Selecione o item *New Foreing Key*. Em *Constraint Name* digite "FK_CONTAS_PAGAR_BANCO". Na coluna *On Field* selecione o campo *CNPJ*. Em *FK Table* devemos escolher a tabela que possui o campo que estamos fazendo referência, nesse caso *Fornecedores*. Em seguida, em *FK Field* devemos escolher os campos da tabela origem que fazem vínculo com os campos escolhidos em *On Field*.

Escolha *CNPJ*, e logo em seguida teremos que configurar apenas mais duas colunas: *Update Mode* e *Delete Mode*. Em ambas, encontramos os valores *No Action*, *Cascade*, *Set Null* e *Set Default*. Para *Update Mode* usaremos *Cascade*, pois toda vez que tivermos alterações na tabela origem, as mesmas devem ser refletidas nas tabelas filhas automaticamente.

Já em *Delete Mode* deixaremos como *No Action*, para que nenhuma ação seja executada caso excluirmos o registro origem. Lembre-se que em nosso sistema não faremos exclusão definitiva no banco de dados, apenas alteramos o *Status* para "I" de inativo.

Feitas essas alterações basta clicar no item do raio. Repita esses passos incluindo mais uma *Foreing Key*, dessa vez incluindo os campos *Banco*, *Agencia* e *Conta* da tabela de Contas a Pagar.

Criando o cadastro de Contas a Pagar

Antes de iniciarmos o desenvolvimento da tela de cadastro de contas a pagar, vamos fazer uma pequena alteração na tela principal do sistema. Vamos incluir um importante componente que vai ajudar a traduzir as principais mensagens de erro exibidas pelo Delphi quando deparado com erros tais como: *Key violation*, valor de campo obrigatório não preenchido etc.

Abra o formulário principal (*frmPrincipal*) e adicione um componente do tipo *ApplicationEvents* ("appEventos") da paleta *Additional*. Esse componente está totalmente ligado ao objeto principal da nossa aplicação, o *Application*.

O *Application* é a classe principal de todo e qualquer projeto gerado pelo Delphi e é o responsável por toda a aplicação. Nele existem alguns eventos que podemos interceptar e desviar para nossas próprias rotinas, facilitando algumas tarefas.

O que faremos agora é fazer tratamentos no evento *OnException* do *appEventos*. Clique duas vezes no evento mencionado e vamos ao código-fonte. Note que o evento recebe dois parâmetros: *Sender* e *E*. O *E* é a exceção gerada por um erro,

sendo nele que devemos nos focar. Esse parâmetro é do tipo *Exception* e recebe várias informações a respeito do erro gerado, uma delas é a mensagem retornada pelo *Message*.

Faremos um *Pos* na mensagem recebida e veremos se ela se encaixa com o que queremos traduzir. Em caso positivo, enviamos a mensagem para o usuário. Vejamos um exemplo na **Listagem 2**.

No código mostrado, procuramos o texto *is not a valid date* dentro da mensagem recebida pela exceção, e caso seja encontrada disparamos a mensagem *Data inválida!*. Dentro do evento do componente criei uma pequena função que recebe a mensagem traduzida e exibe um *MessageDlg* para evitar digitação repetida de código. Incluí alguns dos principais erros que ocorrem com freqüência em aplicativos. Veja a **Listagem 3**.

Ainda no formulário principal precisamos criar um novo botão na barra superior apontando para o cadastro de contas a pagar. Por enquanto apenas inclua um novo botão clicando com o botão direito na *Toolbar* e escolhendo a opção *New Button*.

Em seguida mude o *Caption* para "Contas Pagar". Escolha também uma imagem para ele adicionando-a ao *imgBotoes* criado no artigo anterior. Após criamos a janela de contas a pagar, voltaremos a tela principal para incluir o código de chamada para a tela de lançamentos.

Listagem 2. Exemplo de código de tratamento de exceção

```
if Pos(UpperCase('is not a valid date'), UpperCase(E.Message)) <> 0 then
begin
  Mensagem := 'Data inválida!';
  MostrarMensagem;
end
else
...
```

Listagem 3. Código completo do evento OnException

```
procedure TfrmPrincipal.appEventosException(Sender: TObject; E: Exception);
var
  Mensagem: string;
  Pos1, Pos2: integer;
begin
  Mensagem := '';
  if Pos(UpperCase('is not a valid date'), UpperCase(E.Message)) <> 0 then
    Mensagem := 'Data inválida!'
  else if Pos(UpperCase('must have a value'), UpperCase(E.Message)) <> 0 then
  begin
    Pos1 := Pos('''', E.Message);
    Mensagem := E.Message;
    Delete(Mensagem, Pos1, 1);
    Pos2 := Pos('''', Mensagem);
    Mensagem := Copy(E.Message, Pos1 + 1, Pos2 - Pos1);
    Mensagem := 'O campo ' + Mensagem + ' é de preenchimento obrigatório.';
  end
  else if Pos(UpperCase('key violation'), UpperCase(E.Message)) <> 0 then
    Mensagem := 'Houve violação de Chave. '+'Registro já incluido.'
  else if Pos(UpperCase('Input value'), UpperCase(E.Message)) <> 0 then
    Mensagem := 'Campo preenchido com valor não '+'válido. Proceda a correção.'
  else if Pos(UpperCase('is not a valid time'), UpperCase(E.Message)) <> 0 then
    Mensagem := 'Hora inválida, proceda a correção.'
  else if Pos(UpperCase('Erro ApplyUpdates'), UpperCase(E.Message)) <> 0 then
    Mensagem := 'Não foi possível salvar a '+'modificação no banco de dados.'
  else if Pos(UpperCase('is not a valid float'), UpperCase(E.Message)) <> 0 then
  begin
    Pos1 := Pos('''', E.Message);
    Mensagem := E.Message;
    Delete(Mensagem, Pos1, 1);
    Pos2 := Pos('''', Mensagem);
    Mensagem := Copy(E.Message, Pos1 + 1, Pos2 - Pos1);
    Mensagem := 'O valor  ' + Mensagem + ' não é válido.';
  end
  else if Pos(UpperCase('field value required'), UpperCase(E.Message)) <> 0 then
  begin
    Pos1 := Pos('column ', E.Message) + 7;
    Pos2 := Pos('.', E.Message);
    Mensagem := Copy(E.Message, Pos1, Pos2 - Pos1);
    Mensagem := 'Campo ' + Mensagem + ' deve ser preenchido.';
  end
  else if Pos(UpperCase('FOREIGN KEY'), UpperCase(E.Message)) <> 0 then
    Mensagem := 'Operação não permitida, registro '+'vinculado em outra tabela.';
  if Trim(Mensagem) <> '' then
    MessageDlg(Mensagem, mtWarning, [mbOk], 0);
end;
```

Listagem 4. Código SQL do sqlContasPagar

```
SELECT
  codigo, cnpj, descricao, vlr_real, vlr_pago, juros, mora, dt_cadastro, dt_vecto,
  dt_pagto, banco,  agencia, conta, status, dt_alteracao
FROM
  contas_pagar
WHERE
  status <> 'I' ORDER BY descricao
```

Modificando o Data Module

Para que possamos iniciar o processo de criação da janela de lançamentos, será necessário incluir os componentes de acesso da nova tabela, assim como a criação das funções de pesquisa no cadastro.

Abra o Data Module (*dmdPrincipal*) e adicione um *SQLQuery* ("sqlContasPagar"). Configure sua propriedade *SQLConnection* apontando-a para o *sqlConexao*. Inclua em sua propriedade *SQL* o código SQL da **Listagem 4**.

Clique duas vezes no *sqlContasPagar* e na janela que se abre, clique com o botão direito e selecione *Add all fields*. Em seguida adicione um *DataSetProvider* ("dspContasPagar") e um *ClientDataSet* ("cdsContasPagar") ambos da paleta *Data Access*. Aponte o *dspContasPagar* para o *sqlContasPagar* usando a propriedade *DataSet*. Após, aponte o *cdsContasPagar* para o *dspContasPagar* mudando a propriedade *ProviderName*.

Adicione no *cdsContasPagar* todos os campos da tabela usando o *Fields Editor*. Clique duas vezes no *ClientDataSet* e com o botão direito escolha *Add all fields*. Aproveitando as alterações no Data Module, vamos também incluir outro conjunto de componentes que usaremos para criar uma janela de pesquisa padrão.

Inclua um novo *SQLQuery* ("sqlPesquisaPadrao") e aponte-o para o *sqlConexao* como fizemos anteriormente. Adicione agora um *DataSetProvider* ("dspPesquisaPadrao") e configure a propriedade *DataSet* apontando-o para o *sqlQuery* recém criado. Por último inclua um novo *ClientDataSet* ("cdsPesquisaPadrao") e configure seu *ProviderName*. O Data Module alterado deve se parecer com a **Figura 1**.

> **Nota:** O conjunto de componentes *PesquisaPadrao* não terá campos adicionados, tão pouco instruções SQL, já que faremos isso dinamicamente.

Para completar a alteração, vamos acrescentar as novas funções de pesquisa para que possamos usá-las na janela de lançamentos. Declare as duas funções do código a seguir, na seção *public* do Data Module e pressione CTRL+SHIFT+C para que o Delphi crie o cabeçalho das funções:

```
function PesquisarContasPagar(
  ACampo, AValor: string;
  ATipoPesquisa: Integer): Boolean;
function PesquisarPadrao(ACampo, AValor,
  ATabela: string; ATipoPesquisa: Integer):
Boolean;
```

A explicação para as pesquisas é simples. *PesquisarContasPagar* é semelhante às pesquisas já criadas no capítulo anterior e será usada para localizar lançamentos do contas a pagar. Já *PesquisarPadrao* será usada em conjunto com a caixa de diálogo *frmPesquisaPadrao* que criaremos para auxiliar na busca de fornecedores, clientes e qualquer outro cadastro que precisarmos consultar.

Vá até a seção *implementation* do Data Module e encontre o cabeçalho da função *PesquisarContasPagar,* em seguida, digite o código da **Listagem 5**. Localize agora a função *PesquisarPadrao* e digite o código da **Listagem 6**.

A *PesquisarPadrao* recebe como parâme-

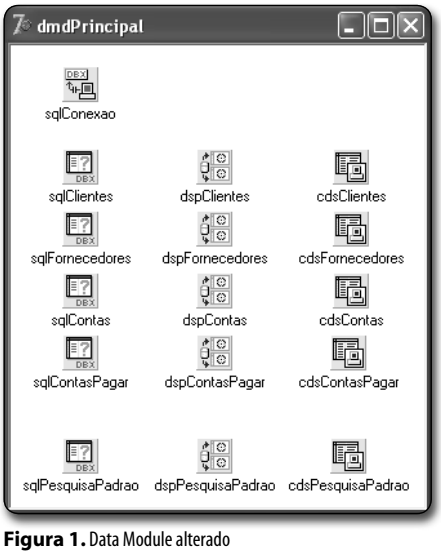


Figura 1. Data Module alterado

Listagem 5. Código da função PesquisarContasPagar

```
function TdmdPrincipal.PesquisarContasPagar(ACampo, AValor: string; ATipoPesquisa: Integer): Boolean;
var
  Operador: string;
  strAux : string;
begin
  case ATipoPesquisa of
    0: Operador := '=';
    1, 2: Operador := 'LIKE';
  end;
  with dmdPrincipal, sqlContasPagar, SQL do
  begin
    cdsContasPagar.Close;
    Clear;
    Params.Clear;
    Add('SELECT');
    Add('  CODIGO, CNPJ, DESCRICAO, VLR_REAL, '+'VLR_PAGO, JUROS, MORA, DT_CADASTRO,');
    Add('  DT_VENCIMENTO, DT_PAGAMENTO, BANCO, '+'AGENCIA, CONTA, STATUS, DT_ALTERACAO');
    Add('FROM');
    Add('  CONTAS_PAGAR');
    Add('WHERE ');
    { Monta a cláusula Where }
    Add('(' + ACampo + ' ' + Operador + ' :' + ACampo + ')');
    cdsContasPagar.FetchParams;
    { A variável strAux foi criada para tratar o valor que está chegando no parâmetro
      AValor. Ele não pode ser maior (em caracteres) do que o tamanho do campo, pois nesse
      caso é gerado uma exceção. }
    case ATipoPesquisa of
  0: cdsContasPagar.Params.ParamByName(ACampo).AsString := AValor;
  1:
    begin
      strAux := Copy(AValor, 1, cdsContasPagar.FieldByName(ACampo).Size-1);
      cdsContasPagar.Params.ParamByName(ACampo).AsString := strAux + '%';
    end;
  2:
    begin
      strAux := Copy(AValor, 1, cdsContasPagar.FieldByName(ACampo).Size-2);
      cdsContasPagar.Params.ParamByName(ACampo).AsString := '%' + strAux + '%';
    end;
end;
Add('  AND STATUS <> ''I'' ');
Add('ORDER BY');
Add('  CODIGO');
cdsContasPagar.Open;
Result := not cdsContasPagar.IsEmpty;
  end;
end;
```

Listagem 6. Código da função PesquisaPadrao

```
function TdmdPrincipal.PesquisarPadrao(ACampo, AValor, ATabela: string; ATipoPesquisa: Integer): Boolean;
var
  Operador: string;
  strAux: string;
begin
  case ATipoPesquisa of
    0: Operador := '=';
    1, 2: Operador := 'LIKE';
  end;
  with dmdPrincipal, sqlPesquisaPadrao, SQL do
  begin
    sqlPesquisaPadrao.Close;
    sqlPesquisaPadrao.SQL.Clear;
    Clear;
    Params.Clear;
    cdsPesquisaPadrao.Close;
    Add('SELECT * FROM ' + ATabela);
    Add('WHERE ');
    Add('(' + ACampo + ' ' + Operador + ' :' + ACampo + ')');
    cdsPesquisaPadrao.FetchParams;
    case ATipoPesquisa of
      0: cdsPesquisaPadrao.Params.ParamByName(ACampo).AsString := AValor;
      1: cdsPesquisaPadrao.Params.ParamByName(ACampo).AsString := AValor + '%';
      2: cdsPesquisaPadrao.Params.ParamByName(ACampo).AsString := '%' + AValor + '%';
    end;
    Add('  AND STATUS = ''A'' ');
    cdsPesquisaPadrao.Open;
    Result := not cdsPesquisaPadrao.IsEmpty;
  end;
end;
```

tro o campo (*ACampo*) e a tabela (*ATabela*) onde será realizada a pesquisa assim como o valor (*AValor*) e o tipo de pesquisa (*ATipoPesquisa*) a ser usada, retornando o resultado da pesquisa.

Para finalizar a alteração, vamos programar os eventos *AfterPost*, *AfterDelete*, *AfterInsert* e *BeforePost* do *cdsContasPagar*. Clique duas vezes sobre o evento *AfterPost* e digite o código a seguir:

```
cdsContasPagar.ApplyUpdates(0);
```

No evento *AfterDelete* apenas associe-o ao evento *AfterPost* escolhendo-o. Já no evento *BeforePost* digite o código a seguir:

```
cdsContasPagar.FieldByName(
  'DT_ALTERACAO').AsDateTime := Now;
```

E por fim vamos digitar o código do evento *AfterInsert* conforme o código da **Listagem 7**.

Esse último, é usado para assumir valores padrão nos campos *Dt_Cadastro*, *Dt_Vecto*, *Vlr_Pago*, *Vlr_Real*, *Juros* e *Mora* como datas e valores monetários logo em seguida do comando *Insert* ou *Append* do *DataSet*.

Listagem 7. Evento AfterInsert do cdsContas_Pagar

```
with cdsContasPagar do
begin
  FieldByName('DT_CADASTRO').AsDateTime := Now;
  FieldByName('DT_VECTO').AsDateTime := Now;
  FieldByName('VLR_PAGO').AsFloat := 0;
  FieldByName('VLR_REAL').AsFloat := 0;
  FieldByName('JUROS').AsFloat := 0;
  FieldByName('MORA').AsFloat := 0;
end;
```

Listagem 8. Botão de pesquisa da caixa de pesquisa padrão

```
procedure TfrmPesquisaPadrao.spbPesquisaClick(Sender: TObject);
begin
  if (Trim(edtPesquisa.Text) = '') or (Length(edtPesquisa.Text) < 4) then
  begin
    MessageDlg('Digite ao menos 4 (quatro) letras '+
      'para realizar a pesquisa.', mtInformation, [mbOk], 0);
    edtPesquisa.SetFocus;
  end
  else
  begin
    with dmdPrincipal do
    begin
      if not (PesquisarPadrao(cbxPesquisa.Items[
        cbxPesquisa.ItemIndex], edtPesquisa.Text, FTabela, cbxTipoBusca.ItemIndex)) then
      begin
        MessageDlg('Nada selecionado para filtro.', mtInformation, [mbOk], 0);
        with dmdPrincipal do
        begin
          sqlPesquisaPadrao.Close;
          cdsPesquisaPadrao.Close;
        end;
      end;
    end;
  end;
end;
```

Listagem 9. Alterando o Caption da janela e carregando o cbxPesquisa

```
Caption := Caption + FTabela;
FDataSetAlvo.GetFieldNames(cbxPesquisa.Items);
cbxPesquisa.ItemIndex := 0;
with dmdPrincipal do
begin
  sqlPesquisaPadrao.Close;
  cdsPesquisaPadrao.Close;
end;
```

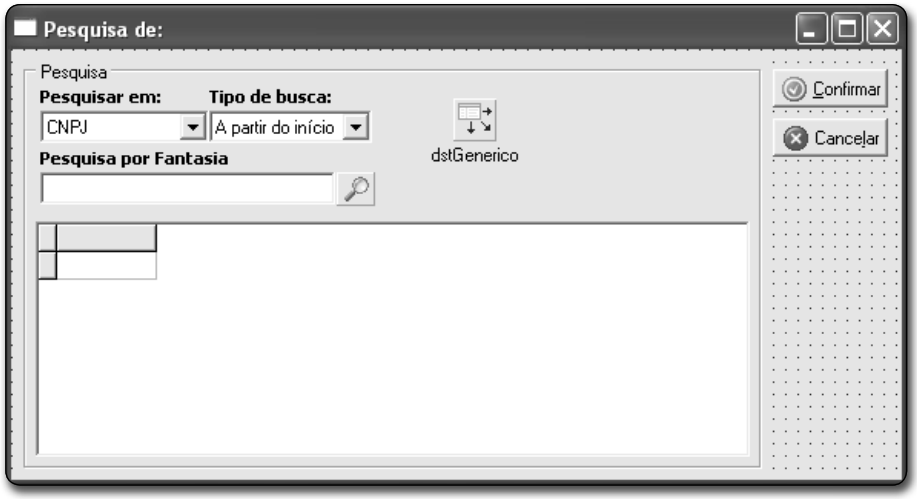


Figura 2. Janela padrão de pesquisa

Criando a pesquisa padrão

Para facilitar o trabalho do usuário final na hora de digitar dados como banco, agência e conta ou selecionar um CNPJ de cliente ou fornecedores, vamos desenvolver uma pequena tela de pesquisa genérica, capaz de localizar registros usando os mesmos moldes do box de pesquisa criados nas telas de cadastro até agora.

Vamos criar um novo formulário clicando em *File>New>Form*. Salve-o como "uPesquisaPadrao.pas" e mude seu *Name* para "frmPesquisaPadrao". Nossa tela deverá ficar como a **Figura 2**.

Para facilitar o desenho da tela, se preferir, copie os campos do box de pesquisa do formulário padrão para dentro da caixa de pesquisa, pois são exatamente os mesmo campos, incluindo os nomes dos componentes.

As únicas alterações efetuadas foram o redimensionamento do *DBGrid* deixando-o mais largo e mais baixo, e retirando todas as colunas do mesmo, já que serão carregadas dinamicamente. Adicione também um *DataSource* ("dstGenerico") e ligue-o ao *ClientDataSet* ("cdsPesquisaPadrao") no Data Module principal. Não esqueça que para isso, você deve adicionar a unit do Data Module ao formulário de pesquisa usando a opção *File>Use unit*. Por fim, ligue o *DBGrid* ao *DataSoure* criado.

Por último, inclua dois *BitBtns* na tela e escolha imagens para eles caso deseje. No primeiro botão, *Confirma*, apenas marque sua propriedade *ModalResult* com a opção *mrOk* e no segundo, *Cancelar*, apenas chame *Close* no evento *OnClick*.

Vamos codificar apenas o botão prin-

cipal de pesquisa e o evento *OnShow* do formulário. Clique duas vezes no botão de pesquisa no centro da janela. Nele adicione o código da **Listagem 8**.

No código anterior, apenas verificamos se o usuário digitou ao menos quatro caracteres para busca e então chamamos a função *PesquisarPadrao* presente no Data Module padrão. No evento *OnShow* do formulário faremos algumas alterações como mudar o *Caption* da janela, atualizar o *cbxPesquisa* com o campos da tabela a ser pesquisada e fechamos os componentes de acesso a dados no data modulo principal. Para isso, digite o código da **Listagem 9**.

Por fim, declare as variáveis *FTabela* do tipo *string* e *FDataSetAlvo* do tipo *TDataSet* na seção *public* do formulário.

Criando a caixa de baixa de lançamentos

Em nosso exemplo, o usuário final poderá incluir lançamentos de contas a pagar como contas de luz, telefone, água, compras diversas etc., como já foi citado. É também nessa tela que poderá ser feita a baixa do lançamento quando o mesmo for efetuado, por esse motivo, se faz necessário a criação de uma pequena caixa de diálogo para receber os valores e efetuar a baixa, o que veremos adiante.

Crie um novo formulário, salvando-o como "uBaixaContasPagar.pas" e dê o nome de "frmBaixaContasPagar". Adicione a unit do Data Module principal usando ALT+F11. Adicione um Data Source ("dstContasPagar") e ligue-o ao *cdsContasPagar* no *dmdPrincipal*.

Em seguida adicione alguns *DBEdits* representando os campos *Vlr_Pago*, *Dt_Pagto*, *Juros*, *Mora*, *Banco*, *Agencia* e *Conta*. Todos eles deverão estar ligados ao *dstContasPagar* e a seus determinados campos pelas propriedades *DataSource* e *DataField*. A sugestão para essa tela pode ser vista na **Figura 3**.

Listagem 10. Botão de pesquisa da caixa de pesquisa padrão

```
uses uPesquisaPadrao;
...
procedure TfrmBaixaContasPagar.sbbSelecionarFornClick(Sender: TObject);
begin
  frmPesquisaPadrao := TfrmPesquisaPadrao.Create(Self);
  with frmPesquisaPadrao do
  begin
    FTabela := 'Contas';
    FDataSetAlvo := dmdPrincipal.cdsContas;
    ShowModal;
    if (ModalResult = mrOk) and not
      (dstGenerico.DataSet.IsEmpty) then
    begin
      with dstContasPagar do
      begin
        DataSet.FieldByName('BANCO').AsString := dstGenerico.DataSet.FieldByName('BANCO').AsString;
        DataSet.FieldByName('AGENCIA').AsString :=
           dstGenerico.DataSet.FieldByName('AGENCIA').AsString;
        DataSet.FieldByName('CONTA').AsString :=
          dstGenerico.DataSet.FieldByName('CONTA').AsString;
      end;
    end;
  end;
  FreeAndNil(frmPesquisaPadrao);
end;
```

Figura 3. Exemplo de lançamentos de contas a pagar

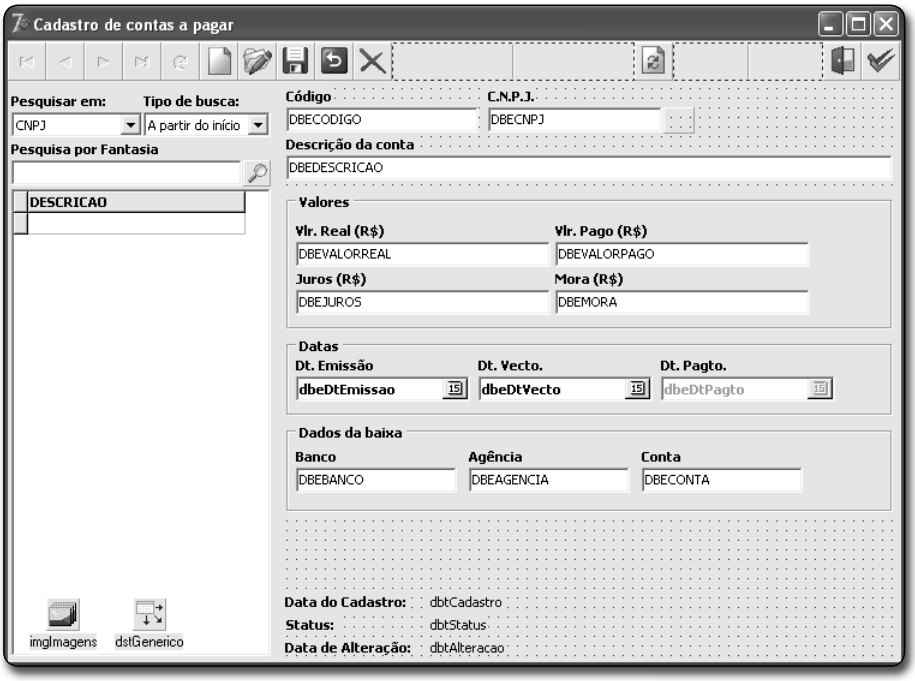


Figura 4. Exemplo de lançamentos de contas a pagar

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Luciano Pimenta, mostrando como instalar os componentes da suíte JEDI.

http://www.devmedia.com.br/articles/viewcomp.asp?comp=3655

Listagem 11. Pesquisando contas a pagar

```
inherited;
if (Trim(edtPesquisa.Text) = '') or (Length(edtPesquisa.Text) < 4) then
begin
  MessageDlg('Digite ao menos 4 (quatro) letras '+
    'para realizar a pesquisa.', mtInformation, [mbOk], 0);
  edtPesquisa.SetFocus;
end
else
begin
  with dmdPrincipal do
  begin
    if not PesquisarContasPagar(cbxPesquisa.Items[
      cbxPesquisa.ItemIndex], edtPesquisa.Text, cbxTipoBusca.ItemIndex) then
      MessageDlg('Nada selecionado para filtro.', mtInformation, [mbOk], 0);
  end;
end;
```

Listagem 12. Código do btnVisualizarTodos

```
inherited;
if MessageDlg('Essa operação poderá demorar alguns '+
  'minutos dependendo do volume de dados.' + #13 +
  'Deseja prosseguir?', mtConfirmation, [mbYes, mbNo], 0) = mrYes then
  with dmdPrincipal do
    if not CarregarTodos(sqlContasPagar, cdsContasPagar, 'CONTAS_PAGAR', 'CODIGO') then
      MessageDlg('Nada selecionado para filtro.', mtInformation, [mbOk], 0);
```

Listagem 13. Botão de pesquisa de fornecedores

```
uses uPesquisaPadrao;
...
procedure TfrmContasPagar.sbbSelecionarFornClick(Sender: TObject);
begin
  inherited;
  frmPesquisaPadrao := TfrmPesquisaPadrao.Create(Self);
  with frmPesquisaPadrao do
  begin
    FTabela := 'Fornecedores';
    FDataSetAlvo := dmdPrincipal.cdsFornecedores;
    ShowModal;
    if (ModalResult = mrOk) and not
      (dstGenerico.DataSet.IsEmpty) then
      frmContasPagar.dstGenerico.DataSet.FieldByName(
        'CNPJ').AsString := dstGenerico.DataSet.FieldByName('CNPJ').AsString;
  end;
  FreeAndNil(frmPesquisaPadrao);
end;
```

Listagem 14. Código do botão de baixa

```
uses uBaixaContasPagar;
...
procedure TfrmContasPagar.tbnBaixaClick(Sender: TObject);
begin
  inherited;
  with dmdPrincipal, cdsContasPagar, frmBaixaContasPagar do
  begin
    if not (FieldByName('STATUS').AsString = 'B') then
    begin
      frmBaixaContasPagar := TfrmBaixaContasPagar.Create(Self);
      Edit;
      dstContasPagar.DataSet.FieldByName(
        'VLR_PAGO').AsFloat := dstGenerico.DataSet.FieldByName('VLR_REAL').AsFloat;
      dstContasPagar.DataSet.FieldByName('DT_PAGTO').AsDateTime := Now;
      frmBaixaContasPagar.ShowModal;
      if frmBaixaContasPagar.ModalResult = mrOk then
      begin
        FieldByName('STATUS').AsString := 'B';
        Post;
      end;
    end
    else
      MessageDlg('Lançamento já baixado!', mtInformation, [mbOk], 0);
    end;
  end;
```

Listagem 15. Chamando o formulário de contas a pagar

```
frmContasPagar := TfrmContasPagar.Create(Self);
try
  frmContasPagar.ShowModal;
finally
  frmContasPagar.Free;
end;
```

Nota: Para facilitar o desenvolvimento da tela, optei por usar componentes da suíte JEDI, como é o caso dos componentes JvDbDateEdit. Eles já possuem as validações necessárias de data e vínculo direto ao DataSet.

Para entender melhor como funcionam esses componentes, consulte o portal DevMedia onde encontram-se diversos artigos falando sobre a suíte. Se desejar, use componentes Edits.

Insira dois botões, onde o primeiro terá apenas a propriedade *ModalResult* alterada para *mrOk*. Em seguida chame o *Close* no evento *OnClick* do segundo botão. Essa caixa de diálogo será chamada do *frmContasPagar* quando acionado o *Baixa*, portanto toda as regras referentes a ela estarão na janela chamadora.

Na tela, codificaremos apenas o botão de pesquisa, a fim de localizar o banco, agência e conta em que foi pago o documento, ou seja, estamos criando um facilitador para o usuário que ao invés de digitar os dados pode simplesmente chamar a caixa de pesquisa padrão e encontrar a conta corrente usada no pagamento. Codifique o evento *OnClick* do botão de pesquisa conforme a **Listagem 10**.

O evento consiste em efetuar uma chamada a *frmPesquisaPadrao*. Nela o usuário poderá localizar a conta corrente no cadastro e confirmar, dessa forma os dados da mesma (banco, agência e conta) serão copiados para os respectivos campos da caixa de diálogo *frmBaixarContasPagar*.

Desenvolvendo a tela de lançamentos

A tela segue o padrão criado no artigo anterior, ou seja, os mesmos botões, o box de pesquisa e até mesmo o tamanho da janela são idênticos. Isso se deve ao fato de usarmos herança para a criação de todas as telas de cadastro em nosso exemplo.

Claro que cada janela possui suas particularidades como mencionado anteriormente, ou seja, na barra superior será adicionado um novo botão, que será usado para efetuar a baixa do lançamento do contas a pagar. A sugestão para essa tela pode ser vista na **Figura 4**.

Ainda observando as particularidades,

vemos um pequeno botão ao lado do campo CNPJ, que será usado para pesquisar o fornecedor a ser usado no lançamento e atualizar o campo de mesmo nome na tabela *Contas_Pagar*.

Vamos iniciar o processo de criação da janela, clicando em *File>New>Other*. Na tela que se abre clique na aba *Form* onde salvamos nosso formulário padrão, então escolha *frmPadrao*, marque a opção *Inherit* e confirme.

> **Nota:** Caso não tenha salvo o formulário padrão no repositório como no artigo anterior, basta acessar a aba com o nome do projeto, nesse caso *SysPague*, e escolher *frmPadrao*. Ambas as formas criam um novo formulário baseado no modelo anterior.

Feito isso salve o formulário como "uContasPagar.pas" e mude seu *Name* para "frmContasPagar". Criada a janela, devemos começar pelas configurações básicas, a primeira consiste em atualizar a propriedade *DataSet* do DataSource (*dstGenerico*) apontando-o para o *cdsContasPagar* no *dmdPrincipal*.

> **Nota:** Não esqueça de adicionar a unit *dmdPrincipal* ao formulário de cadastro usando a opção *File>Use Unit*, ou ALT+F11.

Assim como fizemos nos cadastros de *Clientes*, *Fornecedores* e *Contas Correntes*, vamos codificar os botões de pesquisa no box de pesquisa padrão e na barra de botões. Clique duas vezes no *spbPesquisa* e digite o código da **Listagem 11**. Em seguida, codifique o *btnVisualizarTodos* na barra de botões conforme o código da **Listagem 12**.

Agora é necessário incluir os campos (*DBEdits*) necessários para a visualização dos dados da tabela de Contas a pagar. Insira no formulário alguns *DBEdits* e faça a ligação com seus respectivos campos da tabela usando as propriedades *DataSource* e *DataField* de cada componente.

Como dito anteriormente, optei por utilizar os componentes da suíte JEDI para determinados campos, nesse caso os de Data. Caso não tenha esses componentes instalados no Delphi, adicione *DBEdits* para representar os *Dt_Emisão*, *Dt_Vecto* e *Dt_Pagto*.

Para evitar que o usuário altere indevidamente os campos *Dt_Pagto*, *Banco*, *Agencia* e *Conta* mantenha a propriedade *Enabled* como *False*. Dessa maneira forçamos o usuário ao clicar no *Baixa* e confirmar a baixa do lançamento.

Finalizando o desenvolvimento da janela

Estamos chegando quase ao fim do desenvolvimento da tela de cadastro. Basta codificar o botão para pesquisa de fornecedores e o da baixa. Portanto, insira um *SpeedButton* ("sbbSelecionarForn") para pesquisar o *Fornecedor* e em seu evento *OnClick* digite o código da **Listagem 13**.

O esquema de pesquisa é simples, apenas chamamos a tela de pesquisa padrão informando o *ClientDataSet* alvo e a tabela que será consultada. No final, atualizamos o campo CNPJ da tabela *Contas_Pagar* com o CNPJ do fornecedor encontrado e selecionado.

Vamos ao código do botão de baixa, que por incrível que pareça é bem simples. Apenas verificamos se o registro já não está baixado (*Status = B*) e então chamamos a caixa de diálogo de baixa. O usuário então preenche os campos e confirma. Em seguida o sistema atualiza os campos *Status*, *Vlr_Pago*, *Dt_Pagto*, *Banco*, *Agencia* e *Conta* conforme a janela de baixa. Veja o código da **Listagem 14**.

Pronto, agora basta voltarmos ao formulário principal e incluir a chamada ao cadastro de contas a pagar conforme o código da **Listagem 15**.

Adicione também um novo item de menu, no item *Cadastros* e associe-o ao botão criado na barra. Por último, retire o *frmContasPagar* e os outros formulário criados, da criação automática de formulários em *Project>Options*.

Conclusão

Nesta segunda parte do mini-curso Contas a Pagar e Cobrança vimos como criar a tela de lançamentos de contas a pagar onde o usuário fará a alimentação do sistema. Nas próximas edições veremos como criar outras telas importantes e como evoluir o banco de dados ao longo do desenvolvimento da aplicação. Um forte abraço a todos e até a próxima. ●

+ de 80.000 membros cadastrados
+ de 15.000 exemplos com fontes
+ de 900 apostilas
+ de 4.000 dicas
Fórum Delphi
Artigos
TOTALMENTE GRÁTIS
www.delphi.eti.br
Um dos maiores sites de apoio a desenvolvedores Delphi do Brasil!!!

Impressão de cheques

Como imprimir cheques com as impressoras DP-20 C e DP-20 Plus

Adriano Santos
(artes@doiscliques.com)
é desenvolvedor Delphi desde 1998. Professor e programador PHP. Bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É colunista e membro da Comissão Editorial da revista ClubeDelphi. Mantém o blog Delphi to Delphi (www.delphitodelphi.blogspot.com) com dicas, informações e tudo sobre desenvolvimento Delphi.

Na edição 87 da ClubeDelphi, vimos como desenvolver um aplicativo para emissão de cupom fiscal utilizando a impressora MP-2100 TH FI da Bematech. Seguindo a mesma linha do artigo anterior, veremos como trabalhar em conjunto com outra impressora da linha Bematech: DP-20 C e DP-20 Plus, impressoras de cheques. Veremos como é simples a interação entre Delphi e as impressoras mencionadas.

Este artigo também será de grande utilidade em nosso mini-curso *Sistema de Contas a Pagar e Cobrança*, visto que precisaremos emitir e imprimir cheques no sistema, mais adiante no mesmo. Precisamos entender que existem basicamente dois tipos de impressoras de cheques: com teclado e sem teclado.

Vamos trabalhar neste artigo com as impressoras com teclado. A diferença é que essas, sem teclado, podem ser operadas apenas via software, ou seja, não é possível imprimir cheques diretamente na impressora, pois não há como inserir os dados como valor, favorecido e data.

Em relação as duas impressoras que vamos tratar aqui, a primeira, DP-20 C tem como vantagem, efetuar consultas do cheque, ou seja, na própria impressora podemos checar a autenticidade bem como *status* do cheque como por exemplo se ele foi roubado. Independente disso, ambas funcionam exatamente da mesma forma.

A forma de interação com a impressora se dá por meio de chamadas a sua DDL (*BemaDP32.dll*) disponível gratuitamente para download no site do fabricante. Juntamente com a impressora é disponibilizado um gabarito, para a calibração das coordenadas usadas pela impressora para imprimir todos os dados.

Outros pontos importantes do desenvolvimento são que:

- A DLL deve ser distribuída juntamente com a aplicação;
- Um arquivo (*BemaDP32.ini*), será cria-

do no diretório *C:\Windows* que guardará as configurações de bancos, coordenadas, favorecidos, cidades etc;

• Uma porta COM ou LPT deverá estar disponível para essa impressora;

Neste artigo vamos desenvolver uma pequena aplicação capaz de imprimir em uma dessas impressoras. Mão na massa.

Entendendo o projeto

Para entendermos melhor nosso exemplo vamos a um exemplo prático. Ao abrir o sistema o usuário deverá configurar e abrir a porta de comunicação com a impressora. Isso será feito por meio do menu *Configurações>Configurar e Iniciar porta*.

Esse procedimento poderia ser feito utilizando um arquivo de inicialização (INI) ou qualquer outra forma, mas por questões de didática faremos todos os passos. Após a abertura da porta nosso sistema está pronto para incluir, alterar e/ou excluir bancos do arquivo *BemaDP32.ini* presente no diretório *C:\Windows*.

Incluídos todos os bancos, o usuário poderá tranquilamente, imprimir os cheques no sistema utilizando o menu *Cheques>>Imprimir Cheques*.

Baixando e entendendo a DLL

A primeira providência para iniciar nosso projeto é criar uma pasta para alocar os fontes e baixar os arquivos necessários para o desenvolvimento, nesse caso: a DLL. Para isso, entre no site *www.bematech.com.br* e clique no link *Suporte* no menu superior.

Em seguida localize o menu à esquerda do site e clique em *Downloads e Drivers*. Logo após em *Impressoras de Cheques*. Por fim, selecione o sistema operacional no *ComboBox* a direita do site.

Na lista que se abre, localize o item *DLL de Alto-nível versão 1.2.2.0*, arquivo *bemadp32.zip*, ou digite o endereço: *www.bematech.com.br/suporte/downloads/chq_windows/BEMADP32.zip*. Antes de baixar, crie uma pasta e salve a DLL dentro dela. Adiante usaremos essa mesma pasta para fazer o desenvolvimento do exemplo.

Ao término do download, abra a pasta de destino e descompacte os arquivos que estão no ZIP. Entre os arquivos descompactados estarão:

• *BemaDP32.dll*: responsável pelo envio dos comandos à impressora;

• *BemaDP32.ini*: arquivo de configurações da impressora/DLL tais como bancos, favorecidos etc.;

• *BemaDP32.chm*: help da DLL.

Função	Descrição
Bematech_DP_IniciaPorta	Abre e inicia a porta da impressora.
Bematech_DP_FechaPorta	Fecha a porta de comunicação da impressora.
Bematech_DP_ImprimeCheque	Principal função da DLL, que faz o envio dos comandos de impressão para a DP-20 C ou DP-20 Plus.
Bematech_DP_IncluiAlteraBanco	Incluir/alterar um banco no arquivo *BemaDP32.ini*.
Bematech_DP_ExcluiBanco	Excluir bancos do arquivo *BemaDP32.ini*.

Tabela 1. Principais funções da impressora

Figura 1. Exemplo de tela principal

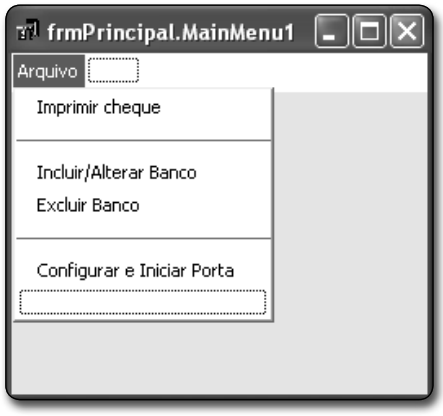


Figura 2. Menu principal da aplicação

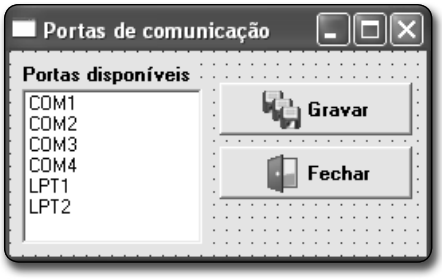


Figura 3. Caixa de configuração de Portas

Figura 4. Caixa de exclusão de bancos

> **Nota:** O arquivo de help contém descrição e exemplo de cada função da DLL, uma excelente fonte de informação para quem está começando.

Pra entendermos melhor a DLL, vejamos algumas de suas principais funções e procedimentos conforme a **Tabela 1**.

Criando o exemplo

Inicie um novo projeto no Delphi utilizando a opção *File>New>Application*. Nomeie o formulário que é criado como "frmPrincipal" e salve-o como "uPrincipal.pas". Em seguida salve o projeto como "Cheques.dpr" usando a opção *File>Save project as*. Vamos criar uma tela principal bastante simples, como visto na **Figura 1**. O *MainMenu* deve ter os itens conforme a **Figura 2**.

Insira também um *Panel* e alinhe-o como *alTop* usando a propriedade *Align*. Dentro dele insira cinco botões com os nomes "spbImprimir", "spbNovoBanco", "spbExcluirBanco", "spbPortas" e "spbFechar", respectivamente da esquerda para a direita. Adicione imagens e aumente a largura de cada botão se necessário.

Vamos adicionar o código para trabalhar com as funções da DLL (semelhante ao que fizemos no artigo de impressora fiscal). Declare o código da **Listagem 1** no formulário antes da seção *implementation*.

Agora, vamos criar a janela de configuração de portas. Crie um novo formulário em *File>New>Form*. Mude seu *Name* para "frmPortas" e salve-o como *uPortas.pas*. Adicione nesse formulário apenas um *Label* com o *Caption* igual a "Portas disponíveis", um *Listbox* ("lstPortas") com os itens "COM1", "COM2", "COM3", "COM4", "LPT1" e "LPT2" adicionados em sua propriedade *Items*. Em seguida insira dois botões no formulário.

A codificação aqui é bastante simples. Basta chamarmos a função *Bematech_DP_IniciaPorta* passando como parâmetro a porta selecionada no *lstPortas*. No primeiro botão (com o *Caption* de "Gravar"), no evento *OnClick*, adicione o código da **Listagem 2**.

> **Nota:** Todas as funções da impressora retornam 0 (zero) para erro e 1 (um) para sucesso além de outros retornos dependendo da função. A variável *Retorno* será declarada no formulário principal como veremos mais adiante.

Agora basta configurar a propriedade *ModalResult* do segundo botão (*Fechar*) para *mrCancel*, assim não precisamos digitar o método de saída no botão. Veja o *layout* do formulário na **Figura 3**.

Faremos o desenvolvimento agora de mais uma tela: Exclusão de Bancos. Crie mais um formulário e salve-o como "uExcluirBanco.pas". Mude seu *Name* para "frmExcluirBanco". Nesse formulário adicione um *Label* e mude seu *Caption* para "Código do Banco" e um *Edit* ("edtBanco").

Em seguida insira dois botões. No primeiro ("btnExcluir"), vamos fazer uma chamada a função *Bematech_DP_ExcluiBanco*, usando o código da **Listagem 3**.

Assim como na tela anterior recebemos o valor na variável *Retorno* e verificamos se a operação foi bem sucedida. Os valores de retorno válidos para essa função são:

- 0 (zero): erro de comunicação;
- 1 (um) : registro apagado com sucesso;
- -2 (menos dois): parâmetro incorreto;
- -3 (menos três): banco não encontrado.

A função *Bematech_DP_ExcluiBanco* exige como parâmetro o código do banco que pode ser *001* para *Banco do Brasil*, e assim por diante (cada banco possui um código único). Pra finalizar, apenas configure o *Fechar* em sua propriedade *ModalResult* assim como fizemos na tela de configuração de portas. Veja o *layout* do formulário na **Figura 4**.

Incluindo bancos no arquivo BemaDP32.ini

O próximo passo é desenvolvermos a tela de inclusão de bancos no arquivo *BemaDP32.ini* presente no diretório *C:\Windows*. Não será necessário criarmos nenhum método de leitura, tão pouco gravação em arquivos INI, pois a própria DLL faz isso através da função *Bematech_DP_IncluiAlteraBanco*.

O que precisamos é criar uma tela de captura das coordenadas, já que a função exige o código do banco e as coordenadas para a configuração. As coordenadas são usadas para que a impressora "saiba" onde deve imprimir os dados do cheque tais como valor, extenso, favorecido e data.

Esse parâmetro precisa ser passado em formato texto (*string*) separado por vírgulas conforme trecho de explicação no arquivo de ajuda da DLL:

Coordenadas: STRING com as coordenadas (linha e coluna) dos campos do cheque. As coordenadas devem ser separadas por vírgula e obedecer a seguinte ordem:

- Linha do valor numérico;

- Primeira linha do valor por extenso;

Listagem 1. Declarando as funções da DLL

```
//Função para abrir a porta de comunicação
function Bematech_DP_IniciaPorta(
  Porta: string): integer; stdcall; far;
  external 'BemaDP32.dll';
//Função para fechar a porta de comunicação
function Bematech_DP_FechaPorta: integer; stdcall;
  far; external 'BemaDP32.dll';
//Função para imprimir o cheque.
function Bematech_DP_ImprimeCheque(Banco: string;
  Valor: string; Favorecido: string; Cidade: string;
  Data: string; Mensagem: string): integer; stdcall;
  far; external 'BemaDP32.dll';
//Função para incluir e alterar o banco e suas
//coordenadas no arquivo bemadp32.ini
function Bematech_DP_IncluiAlteraBanco(Banco: string;
  Coordenadas: string): integer; stdcall; far;
  external 'BemaDP32.dll';
//Função para excluir um banco do arquivo bemadp.ini
function Bematech_DP_ExcluiBanco(
  Banco: string): integer; stdcall; far;
  external 'BemaDP32.dll';
```

Listagem 2. Código para abrir e ativar a porta da impressora

```
procedure TfrmPortas.btnGravarClick(Sender: TObject);
begin
  Retorno := Bematech_DP_IniciaPorta(
    lstPortas.Items[lstPortas.ItemIndex]);
  case Retorno of
    0: MessageDlg('ERRO ao abrir a porta COM. Verifique.', mtError, [mbOk], 0);
     1:
       begin
         MessageDlg('Porta ' + lstPortas.Items[lstPortas.ItemIndex] +
           ' aberta com Sucesso!', mtInformation, [mbOk], 0);
         Close;
       end;
     -2:
   end;
 end;
```

Listagem 3. Excluindo um banco

```
Retorno := Bematech_DP_ExcluiBanco(edtBanco.Text);
case Retorno of
  0: MessageDlg('Erro de Comunicação com a Impressora. '+
       'Verifique.', mtInformation, [mbOk], 0);
  1:
    begin
      MessageDlg('Exclusão efetuada com Sucesso.', mtInformation, [mbOk], 0);
      Close;
    end;
  -2: MessageDlg(
        'Parâmetro de Comando Inválido. Verifique.', mtInformation, [mbOk], 0);
  -3: MessageDlg('Banco Não Encontrado.', mtInformation, [mbOk], 0);
end;
```

- *Segunda linha do valor por extenso;*
- *Linha do favorecido;*
- *Linha da cidade;*
- *Coluna do valor numérico;*
- *Coluna da primeira linha do valor por extenso;*
- *Coluna da segunda linha do valor por extenso;*
- *Coluna do favorecido;*
- *Coluna da cidade;*
- *Coluna do dia;*
- *Coluna do mês e;*
- *Coluna do ano."*

Também é importante lembrar que a impressora possui um gabarito de coordenadas que é enviado junto com o equipamento no ato da aquisição, também conforme instruções no arquivo de ajuda.

Bem, vamos ao desenvolvimento da janela de inclusão de bancos. Crie um novo formulário e salve-o como "uConfiguraBanco.pas". Em seguida mude seu *Name* para "frmConfiguraBanco". Nossa tela deverá se parecer com a **Figura 5**.

Como pode ver no canto superior esquerdo foi incluído um *GroupBox* e nele um *Edit* ("edtBanco"). Logo em seguida um novo *GroupBox*, onde foram adicionados 13 (treze) *Edits* para receber as coordenadas de cada campo do cheques.

De cima para baixo e da esquerda para a direita nomeie o campos com os respectivos nomes: "edtLinhaNum": linha do valor numérico; "edtLinhaExt1": primeira linha do valor por extenso; "edtLinhaExt2": segunda linha do valor por extenso; "edtLinhaFavorecido": linha do favorecido; "edtLinhaCidade": linha da cidade; "edtColNum": coluna do valor numérico; "edtColExt1": coluna da primeira linha do valor por extenso; "edtColExt2": Coluna da segunda linha do valor por extenso; "edtColFavorecido": coluna do favorecido; "edtColCidade": coluna da cidade; "edtColDia": coluna do dia; "edtColMes": coluna do mês e "edtColAno": coluna do ano.

A codificação nessa tela é bastante simples como nas telas anteriores. Primeiramente guardamos os valores de todas as coordenadas em uma variável do tipo *string* (chamada *Coordenadas*).

Cada coordenada é adicionada separada por vírgula como indica o arquivo de help. No final teremos uma *string* mais ou menos como no código a seguir:

```
Coordenadas = '2,2,3,4,2,5,2,3,6,2,5,1,2';
```

Em seguida chamamos *Bematech_DP_IncluiAlteraBanco* passando como parâmetro o banco e a variável *Coordenadas*. Seguindo a mesma lógica empregada nas funções anteriores, recebemos o retorno da função na variável de mesmo nome. Caso o retorno seja 1 (um) enviamos a mensagem ao usuário informando

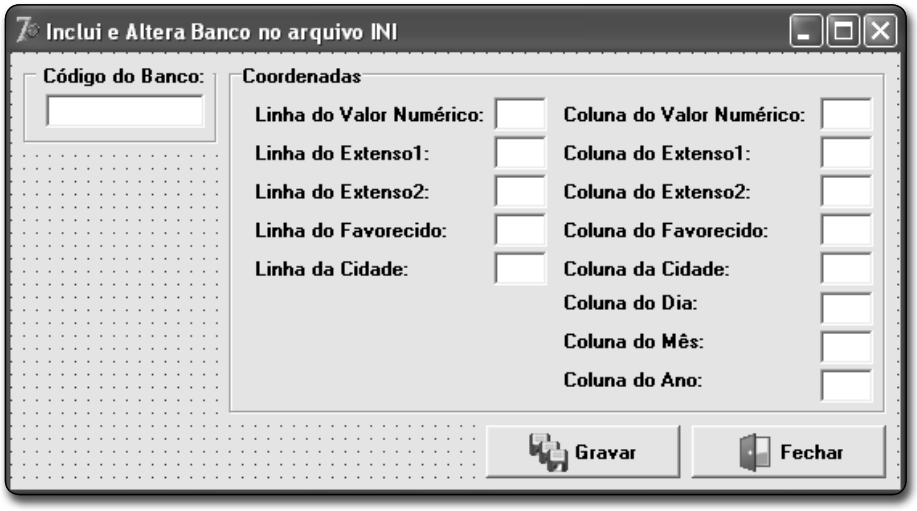


Figura 5. Caixa de inclusão/alteração de bancos

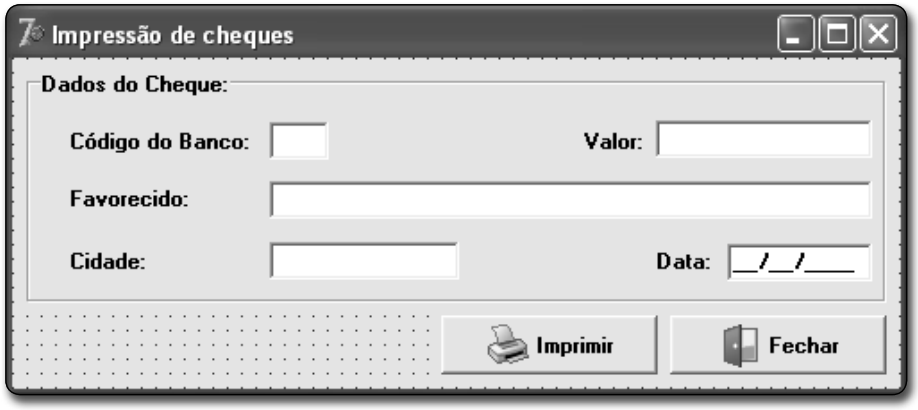


Figura 6. Impressão de cheques pelo sistema

que o banco foi alterado/inserido com sucesso.

> **Nota:** Caso já exista o banco cadastrado no arquivo *BemaDP32.ini* o mesmo será alterado automaticamente pela DLL.

Vejamos na **Listagem 4** o código completo do *Gravar* da janela em questão.

Na **Listagem 5** um trecho do arquivo *BemaDP32.ini* já com alguns bancos adicionados.

Imprimindo cheques

Nossa última providência agora é criar a tela de impressão de cheques. É possível notar que a rotina é bastante simples de implementar, assim como todas as telas já criadas até aqui. Crie um novo formulário usando *File>New>Form* e salve-o como "uImprimir.pas". Modifique seu *Name* para "frmImprimir" e adicione alguns componentes como na **Figura 6**.

Basicamente, a tela fará a captura dos dados digitados nos *Edits* em tela e enviará para a função *Bematech_DP_ImprimeCheque* da *BemaDP32.dll*. Os parâmetros requeridos são: banco, valor, favorecido, cidade, data e mensagem (que não usaremos).

Assim como vimos em todos os casos, chamamos a função da DLL atribuindo seu retorno a variável de mesmo nome. Se o *Retorno* for igual a 1 (um) significa que tudo correu bem e o comando foi enviado com sucesso para a impressora.

Listagem 4. Código de gravação do banco no arquivo BemaDP32.ini

```
procedure TfrmConfiguraBanco.btnGravarClick(Sender: TObject);
var
  Coordenadas: string;
begin
  Coordenadas :=
    edtLinhaNum.Text  + ',' + edtLinhaExt1.Text + ',' +
    edtLinhaExt2.Text + ',' +
    edtLinhaFavorecido.Text + ',' +
    edtLinhaCidade.Text + ',' +
    edtColNum.Text + ',' + edtColExt1.Text + ',' +
    edtColExt2.Text + ',' +
    edtColFavorecido.Text + ',' +
    edtColCidade.Text + ',' + edtColDia.Text + ',' +
    edtColMes.Text + ',' + edtColAno.Text;
  Retorno := Bematech_DP_IncluiAlteraBanco(edtBanco.Text, Coordenadas);
  case Retorno of
    0: MessageDlg('Erro de Comunicação com a '+
      'Impressora. Verifique.', mtInformation, [mbOk], 0);
    1:
      begin
        MessageDlg('Programação efetuada com '+'Sucesso.', mtInformation, [mbOk], 0);
        Close;
      end;
    -2: MessageDlg('Parâmetro de Comando Inválido. '+'Verifique.', mtInformation, [mbOk], 0);
  end;
end;
```

Listagem 5. Exemplo de arquivo BemaDP32.ini

```
[Formato]
001=3,2,1,2,1,7,2,2,3,5,2,5,2
033=2,3,4,5,6,7,8,9,4,7,8,7,3
341=3,3,1,2,2,7,2,5,5,5,2,2,2
237=4,2,4,2,1,1,2,2,3,5,4,5,1
```

Listagem 6. Código de impressão de cheques

```
procedure TfrmImprimir.btnGravarClick(Sender: TObject);
var
  Banco, Valor, Favorecido, Cidade, Data: string;
begin
  Banco := edtBanco.Text;
  Valor := edtValor.Text;
  Favorecido := edtFavorecido.Text;
  Cidade := edtCidade.Text;
  Data := mskData.Text;
  Retorno := Bematech_DP_ImprimeCheque(Banco, Valor, Favorecido, Cidade, Data, PChar(''));
  case Retorno of
    0: MessageDlg('Erro de Comunicação com a '+
         'Impressora. Verifique.', mtInformation, [mbOk], 0);
    1:
      begin
        MessageDlg('Os dados foram enviados com '+'sucesso.', mtInformation, [mbOk], 0);
        Close;
      end;
    -2: MessageDlg('Parâmetro Inválido. Verifique.', mtInformation, [mbOk], 0);
    -3: MessageDlg('Banco não encontrado no arquivo '+
          'de bancos. Verifique.', mtInformation, [mbOk], 0);
  end;
end;
```

Todos os parâmetros da função, incluindo *Valor* e *Data*, são do tipo *string*, ou seja, é altamente recomendável que se façam os devidos tratamentos nas variáveis para evitar erros de compilação.

Em nosso exemplo, estamos enviando o conteúdo do *edtBanco*, ou seja, sua propriedade *Text* que já é do tipo *string*. Veja na **Listagem 6** o código completo do evento *OnClick* do *Imprimir* (preste atenção ao nome dos componentes de tela).

O evento consiste em capturar o conteúdo de cada *Edit* e alimentar suas respectivas variáveis. Em seguida enviamos as variáveis para a função através dos parâmetros.

Finalizando e testando o exemplo

Devemos incluir no formulário principal, todas as units das janelas criadas utilizando a opção ALT+F11. Em seguida faça as chamadas dos formulários para que possamos iniciar a inclusão de bancos, configuração de porta e impressão dos cheques. O código completo para a chamada de cada formulário encontra-se na **Listagem 7**.

Antes de testar, precisamos declarar a variável que recebe o retorno das funções. Abaixo da seção *var* do formulário, declare a variável, usando o seguinte código:

```
var
  frmPrincipal: TfrmPrincipal;
  Retorno: Integer;
```

Por fim, precisamos adicionar a unit do formulário principal, em todos os formulários que fazem chamada às funções da DLL. Compile o projeto e execute-o. Logo que o sistema abrir, precisamos configurar e abrir a porta de comunicação.

Para isso, clique no *Configurar Porta* e escolha uma das portas disponíveis. Caso a porta escolhida retornar sem erros, uma mensagem de confirmação será mostrada assim como codificamos anteriormente. Se a porta retornar problemas tente outra até que uma delas retorne sem erros.

Em seguida insira alguns bancos e então imprima os cheques. Para não deixar a comunicação "aberta" com a impressora de cheques, precisamos chamar a função da DLL. No evento *OnDestroy* utilize o código da **Listagem 8**.

Considerações Finais

Além das funções utilizadas neste exemplo, existem outras como, por exemplo, *Bematech_DP_CancelaRelatorio* que cancela o relatório emitido automaticamente a cada 432 cheques impressos manualmente.

Além disso, podemos imprimir cópias de cheque usando o comando *Bematech_DP_ImprimeCopiaCheque* ou ainda enviar ou formatar textos usando as funções *Bematech_DP_ImprimeTexto* e *Bematech_DP_FormataTexto* respectivamente. Tais comandos são interessantes quando é necessário limpar o *buffer* de impressão.

Uma boa fonte de aprendizado está contida no arquivo *BemaDP32.chm*, arquivo de help citado no início do artigo. Ele contém informações valiosas sobre a DLL e suas funções incluindo também exemplos que podem ser facilmente aproveitados. Veja na **Figura 7** o projeto em execução.

Conclusão

Neste artigo vimos de forma simples como interagir com impressoras de cheques DP-20 C e DP-20 Plus da empresa Bematech. No passo a passo, aprendemos a inicializar a porta de comunicação, incluir, alterar e excluir bancos nos arquivos de configuração das impressoras e ainda fazer a impressão em si.

É importante lembrar que existem outros fabricantes de impressoras de cheques e o método empregado aqui pode sofrer alterações de fabricante para fabricante. Um forte abraço a todos e até a próxima. ●

Links

Site do fabricante das Impressoras de Cheques DP-20 C e DP-20 Plus

www.bematech.com.br

Listagem 7. Código de chamada dos formulários do sistema

```
procedure TfrmPrincipal.sbpChequeClick(Sender: TObject);
begin
  frmImprimir := TfrmImprimir.Create(Self);
  try
    frmImprimir.ShowModal;
  finally
    FreeAndNil(frmImprimir);
  end;
end;
procedure TfrmPrincipal.spbNovoBancoClick(Sender: TObject);
begin
  frmConfiguraBanco := TfrmConfiguraBanco.Create(Self);
  try
    frmConfiguraBanco.ShowModal;
  finally
    FreeAndNil(frmConfiguraBanco);
  end;
end;
procedure TfrmPrincipal.spbExcluirBancoClick(Sender: TObject);
begin
  frmExcluirBanco := TfrmExcluirBanco.Create(Self);
  try
    frmExcluirBanco.ShowModal;
  finally
    FreeAndNil(frmExcluirBanco);
  end;
end;
procedure TfrmPrincipal.spbPortaClick(Sender: TObject);
begin
  frmPortas := TfrmPortas.Create(Self);
  try
    frmPortas.ShowModal;
  finally
    FreeAndNil(frmPortas);
  end;
end;
```

Listagem 8. Fechando a comunicação com a impressora de cheque

```
Retorno := Bematech_DP_FechaPorta();
if Retorno = 0 then
  MessageDlg('Problemas ao tentar liberar a Porta. '+
    'Verifique.', mtInformation, [mbOk], 0)
else
  MessageDlg('Porta Serial Liberada.', mtInformation, [mbOk], 0);
Application.Terminate;
Exit;
```

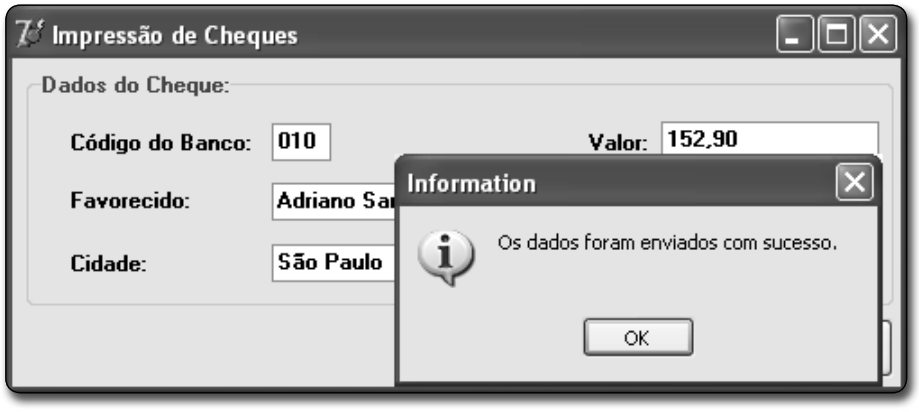
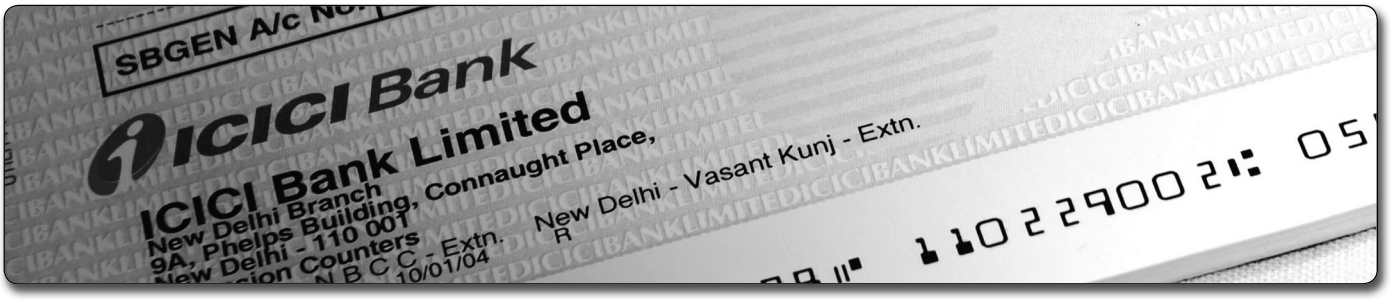


Figura 7. Projeto de impressão de cheques



Versões de Delphi

Saiba qual versão do Delphi seus projetos precisam

Fabrício Desbessel
(fabricio@fabricio.pro.br)
é professor de Linguagem de Programação do Curso Técnico em Informática do Colégio Frederico Jorge Logemann de Horizontina/RS e da FAHOR Faculdade Horizontina. Delphiano de coração está sempre disposto a provar que com o Delphi sempre teremos a melhor solução. Site www.fabricio.pro.br.

Para escrever este artigo foi necessário recordar o meu passado, buscando toda a caminhada em programação e em especial com o Delphi. No início de minha faculdade (1996) aprendi Algoritmos. Como era difícil entender o tal *Português Estruturado* onde precisávamos imaginar o comportamento dos programas. Junto com Algoritmo aprendi a programar em Turbo Pascal 5 e tudo ficou mais fácil, pois podíamos testar na prática.

Ao mesmo tempo, eu e meu colega, estávamos assumindo o setor de informática em uma escola. Era necessário escolher um IDE (*Integrated Development Environment*) e uma linguagem de programação para informatizarmos a escola bem como lecionarmos no curso técnico em processamento de dados que existia na época.

Lembro da primeira vez que vi o Delphi 1. Parecia mágica arrastar componentes, configurar propriedades e codificar eventos. Isso tudo no tempo do Windows 3.11.

Mesmo assim a tarefa de escolher era difícil, pois afetaria até o currículo do curso. Muito se pensava na seqüência que o IDE teria. Junto com o Delphi analisávamos outros produtos como o V.O. (*Visual Object*) que não existe mais e também o Fox Pro. Acabamos escolhendo o Delphi e fazendo um treinamento já no Delphi 2. Bem, no final, até hoje utiliza-se Delphi nessa escola e olha que já se passaram 11 anos.

Acredito que para escolher uma ferramenta de trabalho, devemos levar em consideração vários fatores e, principalmente, o histórico do produto. O modismo da atualidade não deve ser considerado. Todo novo produto deve ter um tempo de aceitação e comprovação de suas funcionalidades. O tempo ideal, não sei. O que sei é que se uma empresa tem um produto bom, com mais de 10 anos, terei a confiança de utilizar qualquer produto dessa empresa.

Nessa caminhada a Borland, visando ações estratégicas fez algumas mudanças como podemos citar o caso da Inprise e

agora a CodeGear. Nesse último, achei correto dividir as empresas que possuem focos diferentes. Só não achei justo o novo nome ficar com os produtos de desenvolvimento, visto que eles são os mais antigos e que mantiveram a Borland até os dias atuais.

No passado, sabíamos que deveríamos utilizar sempre a última versão do Delphi que era a mais atualizada tecnologicamente. Tínhamos diferença entre *Professional*, *Enterprise* e *Architect* na questão de restrições de tecnologias que não estavam disponíveis. Mas, nada se compara aos dias atuais onde temos várias versões e sabores de Delphi. Na seqüência deste artigo vamos tentar abordar as opções de Delphi atuais, buscando identificar as melhores para cada caso de uso.

Delphi 7

Alguns chamam essa versão como a "imortal". Lançada em 2002 é a mais utilizada até o momento, traz todos os recursos necessários para o desenvolvimento de sistemas conectados a banco de dados, aplicações multicamadas, aplicações para Internet, aplicações multimídia, aplicações educativas, aplicações gráficas, etc.

Aliás, esse grande leque de tipos de aplicação é um grande trunfo do Delphi frente aos concorrentes. Também é importante citar a compatibilidade com versões anteriores. Isso parece uma coisa tão simples, mas existem muitas ferramentas que você não consegue abrir e executar projetos de versões anteriores sem fazer mudanças no código.

Já ouvi o comentário que as versões com números ímpares sempre foram as melhores do Delphi. Talvez aí esteja o segredo do Delphi 7, que até o momento é o mais utilizado pela comunidade de desenvolvedores.

É importante ressaltar o crescimento que o Delphi teve a cada versão. Para tanto, vamos a um resgate histórico, apresentando algumas novidades de cada versão:

• Delphi 1 (1994): Surge a Linguagem *Object Pascal* com o primeira IDE totalmente RAD (*Rapid Application Development*) baseada em formulários e Orientadas a Objetos. Com um compilador rápido, ótimo suporte a banco de dados e com uma tecnologia de componentes a ferramenta revolucionou o mercado;

• Delphi 2 (1995): Delphi surge como uma das primeiras ferramentas de desenvolvimento compatíveis com o Windows (Win32). A linguagem evoluiu oferecendo novos tipos de dados como o *Variant* e incorporando componentes de gráficos, e cubos de decisão;

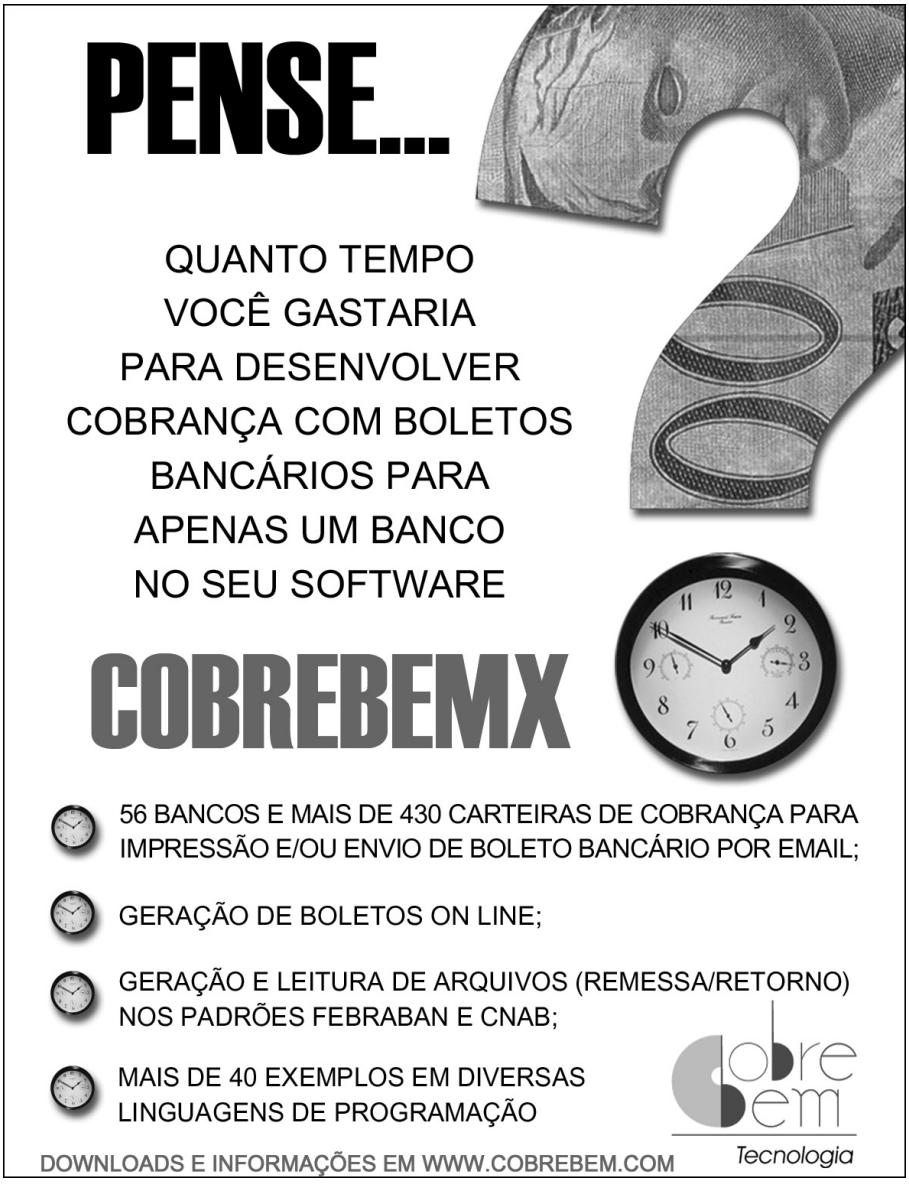
• Delphi 3 (1997): Delphi apresenta a tecnologia WebBroker para desenvolvimentos de aplicações para a Internet (CGI e ISAPI/NSAPI), o *Midas* para o desenvolvimento de aplicações multi-camadas;

• Delphi 4 (1998): Veio com um suporte melhor à OLE e COM, extensões de componentes de banco de dados e muitas adições nas classes básicas da VCL para acompanhar a versão atual do Windows (98);

• Delphi 5 (1999): Apresentou um grande aperfeiçoamento no IDE, mais suporte de dados estendidos com ADO e Interbase e uma versão aperfeiçoada do *Midas* com suporte a Internet. Nessa versão aparece a primeira integração com uma ferramenta de gerenciamento de projeto (TeamSource);

• Delphi 6 (2001): Novamente a Borland revolucionou apresentando um suporte ao desenvolvimento de aplicações independentes de plataforma com a tecnologia CLX (*Component Library for Cross-Platform*). Outra grande inovação é o mecanismo de acesso a dados dbExpress;

• Delphi 7 (2002): Apresentou-se como uma ferramenta mais robusta na utilização de todas suas tecnologias. Ofereceu várias ferramentas de terceiros como Rave (relatórios) IntraWeb (aplicações Web) e ModelMaker (UML).



O importante é que a cada versão tivemos melhorias nas tecnologias existentes, bem como incorporação de novas tecnologias. Nesse sentido, a Borland sempre saiu na frente. O Delphi, além de ter uma ótima linguagem de programação, tem no seu IDE uma interface intuitiva, onde o aprendizado se torna prazeroso e, de certo modo, muito fácil.

Achei importante colocar todos esse contexto histórico, pois acho que é fundamental avaliar todo o crescimento que o Delphi apresentou durante os anos. Com esse contexto fica fácil projetar o futuro e saber até onde o Delphi pode chegar.

Delphi 8 e Delphi 2005

O Delphi 8 foi o primeiro passo da Borland na plataforma .NET. Podemos dizer que a Borland foi inovadora nesse sentido, pois apresentou o melhor IDE para desenvolvimento de aplicações em .NET, tanto para desktop como para Web.

O Delphi 2005 integrou no mesmo IDE (chamada de BDS) o desenvolvimento de aplicações .NET e Win32, dando a possibilidade de desenvolvimento em outras linguagens como C++ Builder e C#. Possibilitou o desenvolvimento Web utilizando o framework ASP.NET.

Delphi 2006

No final de 2005, recebemos o lançamento do Delphi 2006. Utilizando o IDE chamada BDS (*Borland Developer Studio*), versão 4, traz a possibilidade de utilização de Delphi for .NET, Delphi for Win32, C++ Builder e C#. Entre as várias novidades podemos citar os recursos de edição de código como o *Refactoring*, *Live Templates*, *Block Completion*, *History Tab*, entre outros, que agilizam o desenvolvimento das aplicações, além de facilitar a vida do desenvolvedor.

O Delphi 2006 é comercializado em três edições:

Borland® Delphi® 2006

KEY FEATURE HIGHLIGHTS*	d = Delphi (Win32) support D = Delphi (.NET) support	# = C# support C = C/C++ support		
VCL		ARC	ENT	PRO
New Live Designer Guidelines with "smart" snap to drag lines, margins, padding, and text-based alignment	dDC	■	■	■
New VCL Flow Panel and Grid panel for HTML style and Table style form layouts with Autopositioning	dDC	■	■	■
IDE				
New Live Templates System - tab through fields and insert points of any template	dD#C	■	■	■
New "Smart" Block completion - anticipates and gracefully adds block closures as you type	dD#C	■	■	■
DATABASE DEVELOPMENT AND ACCESS				
New! Database Explorer with dbExpress and drag-n-drop support	dDC	■	■	■
Latest dbExpress (for VCL) and BDP ADO.NET (for WinForms/ASP.NET) client/server and multi-tier drivers for Borland® Interbase® 7.5, Oracle® 10g, IBM® DB2® v8.x, Microsoft® SQL Server™ 2000/2005, Informix® v9.x, SQL Anywhere® v9, MySQL® v4.0.24, and Sybase® v12.5	dD#C	■	■	
New! BDP ADO.NET Connection Pooling and SQL Tracing	D#	■	■	■
BORLAND ECO™ III RAPID DEVELOPMENT				
New! ECO State Machines powered by State Chart visual diagrams	D#	■		
New! Multiple database support, simultaneous persistence to multiple and mixed database servers	D#	■		
New! Select ECO base features in Pro and Enterprise editions	D#	■	■	■
TEAM PRODUCTIVITY AND SOURCE CODE MANAGEMENT				
Complete native Borland® StarTeam® and Borland® CaliberRM™ IDE Integration	dD#C	■	■	■
INTRAWEB V8.0				
WYSIWYG RAD Web page development	dDC	■	■	■
WYSIWYG RAD Web application development	dDC	■	■	
ASP.NET				
NEW HTML/ASP.NET source preservation	D#	■	■	■
Enhanced HTML/ASP.NET Tag Editor	D#	■	■	■
New ASHX HTTP Handler Wizard	D#	■	■	■
BORLAND® TOGETHER® MODELING				
Full two-way LiveSource class diagrams	dD#	■	■	■
UML diagrams and design pattern support	dD#	■	■	■
Integrated audits and metrics	dD#	■	■	
Documentation generation	dD#	■	■	

* Full Features Matrix available at www.borland.com

Figura 1. DataSheet do Delphi 2006

• *Delphi 2006 Professional*: ideal para profissionais autônomos que desenvolvem aplicações desktop ou web que conectam em banco de dados locais e aplicações que utilizam o ECO III com mapeamento objeto-relacional transparente em arquivos XML.

Características:

- Desenvolvimento RAD com suporte às linguagens de programação Delphi, C, C++ e C#;
- Cliente visual WYSIWYG e desenvolvimento de aplicação Web
- Conectividade local a base de dados para InterBase, dBase, Paradox;
- ADO, MySQL, MSDE e Access;
- Funções básicas de mapeamento objeto-relacional e de persistência transparente de objeto XML local suporte Win32, .NET e ASP.NET.

• *Delphi 2006 Enterprise*: ideal para utilização de organizações que desenvolvem aplicações de pequeno a médio porte, podendo ser de missão crítica e conectados a banco de dados remotos ou locais, de alta performance. Oferece a tecnologia IntraWeb para desenvolvimento de aplicações para Internet de forma rápida.

Características:

- Modelagem Together Integrada com UML, *LiveSource*, Estatísticas e Auditorias integradas, e Geração de Documentos;
- Conectividade a servidor de base de dados de alta performance para InterBase, SQL Server, Oracle, IBM DB2, MySQL e outros;
- Mapeamento objeto-relacional e persistência transparente de base de dados;
- Desenvolvimento de aplicações de base de dados multicamadas Win32 e . NET;
- Desenvolvimento de aplicação Web IntraWeb RAD;
- Borland StarTeam Standard Edition.

• *Delphi 2006 Architect*: apresenta todos os recursos da Enterprise e todo o framework ECO III com diagramas de estados e persistência simultânea de dados em servidores de banco de dados múltiplos e distintos.

Características:

- Diagramas State Chart;
- ECO State Machines Executáveis;
- Suporte, multi e misto, a base de dados ECO.

Veja mais sobre as diferenças de edições na **Figura 1** que apresenta o *DataSheet* da versão 2006.

Depois de todas essas características é que vem a grande pergunta: Qual edição comprar? Para responder essa pergunta devemos pensar na real utilização da ferramenta. Se você é um desenvolvedor iniciante, ou que trabalha somente com pequenas empresas que não necessitam de um servidor de banco de dados dedicado, a edição ideal é a *Professional*.

Agora se seus projetos requerem um banco de dados de grande porte ou, precisa modelar a aplicação em UML, ou ainda precisa desenvolver uma aplicação Web de forma muito rápida, a edição ideal é a *Enterprise* que também possibilitará uma integração com o StarTeam para gerenciamento do projeto.

A edição *Architect* é para empresas que

efetivamente farão uso do ECO III para ganhar performance e produtividade em aplicações que exigem muita mudança e escalabilidade. Na minha opinião, poucas empresas utilizam esse framework o que não justifica a aquisição dessa edição. Portanto, a edição mais indicada para uma empresa de desenvolvimento, no meu ponto de vista, é a *Enterprise*.

Turbo Delphi

Como o Delphi incorporou várias tecnologias e linguagens, em um único IDE o seu preço ficou bem alto, o que resultou em várias reclamações para a Borland que, de forma brilhante traçou uma nova estratégia, criando as versões Turbo.

O Turbo é uma versão onde você pode escolher a linguagem de desenvolvimento. Por exemplo, se só desenvolvo aplicações para Web utilizando Dephi for .NET, não tenho porque comprar todo o Delphi 2006.

Com essa estratégia você pode escolher o que comprar e pagar somente pela tecnologia que precisa. Agora veja os produtos Turbo:

- *Turbo Delphi*: desenvolvimento de aplicações Win32;
- *Turbo Delphi* .NET: desenvolvimento de aplicações .NET;
- *Turbo C++*: desenvolvimento de aplicações Win32;
- *Turbo C#*: desenvolvimento de aplicações .NET com C#.

Além da separação das linguagens que resultou em um preço mais justo pelo produto, a Borland preocupou-se como os desenvolvedores iniciantes, que estão aprendendo a utilizar os produtos e criou uma edição dos produtos chamada *Explorer*.

As edições *Turbo Explorer* diferem na característica de acessarem somente banco de dados locais e, não permitirem a adição de novos componentes em seu IDE. Mesmo assim, é uma ótima opção para o aprendizado das linguagens e tecnologias.

As versões Turbo, no seu contexto, são perfeitas para empresas que utilizam uma única linguagem de programação e que não pretendem pagar por tecnologias que não serão utilizadas pela sua equipe.

Delphi® 2007 for Win32®

The RAD visual development environment for Windows®

SDKS	PRO	ENT
Microsoft Win 32 SDK with support for Windows Vista including themed applications, glassing, file dialogs, and Task Dialog components	■	■
VCL		
VCL for the Web — complete RAD Web application and Web page development with AJAX support		■
VCL for the Web —Web page development with AJAX support	■	■
DATABASE DEVELOPMENT AND ACCESS		
DBX4 database architecture to streamline connectivity, simplify the writing of data drivers and supports connection pooling	■	■
Database drivers for Interbase 2007, MySQL 4.1 and 5.0, and SQLAnywhere 9	■	■
Database drivers for Oracle 10g, DB2 8.x, MS SQL Server 2000/2005, Informix 9x, and Sybase 12.5		■
PROJECT MANAGEMENT		
Powerful build system using MS Build including support for pre and post-build events and named build configurations	■	■
TOGETHER® MODELING		
Together modeling program with full two-way LiveSource™ class diagrams, OCL 2.0 support, "UML in Color" profile, Diagram hyperlinking and annotations, and Diagram printing	■	■
Expanded Together modeling diagrams and charts including Sequence diagrams, Collaboration diagrams, State Charts, Deployment diagrams, Use Case diagrams, Activity diagrams, and Component diagrams		■
Additional Together modeling features including Design Patterns support, XMI 1.1 Import/Export, Documentation generation, and Audits and metrics		■
DEVELOPER SUPPORT		
Updated TeeChart Version 7 – library of charting and plotting components and controls	■	■
Updated Indy Version 10 - an open source internet component suite comprised of popular internet protocols written in Delphi and based on blocking sockets	■	■
Rave Reports Update – a powerful and flexible suite of visual reporting components and classes	■	■

Figura 2. DataSheet do Delphi 2007

Delphi 2007

A nova versão do Delphi para aplicações Win32. A Borland, através da sua subsidiária CodeGear ouviu a comunidade de desenvolvedores que solicitavam uma continuação do Win32 e lançou a versão sucessora do Delphi 7. Entre as características destaco a compatibilidade com o Windows Vista e AJAX, deixando suas aplicações em dia com as tecnologias atuais.

O Delphi 2007 é a versão recomendada para o desenvolvimento de aplicações Win32, substituindo o Delphi 7. Vários desenvolvedores questionam se devem ou não fazer a troca e eu lhes respondo: trocaria só pelas facilidades encontradas no editor de código como *Live Templates*, *Block Completion* etc.

Essas facilidade parecem pequenos detalhes, mas resultam em ganho de performance no desenvolvimento das aplicações. Outro atrativo dessa versão é que nossas aplicações estarão compatíveis com o Windows Vista, o que deve ser uma preocupação, uma vez que as máquinas novas estão saindo das fábricas com essa versão do Windows. Portanto é hora de mudar. Avance e comece a utilizar o novo Delphi 2007.

O Delphi 2007 é comercializado em duas edições: *Professional* e *Enterprise*. A diferença entre as edições está na possibilidade de utilização da VCL para Web (IntraWeb) e na integração com o Together. Veja na **Figura 2** o *DataSheet* do produto.

Delphi for PHP

Essa é mais uma novidade da CodeGear. Considero brilhante a idéia de oferecer uma ferramenta totalmente RAD para o desenvolvimento de aplicações Web em PHP. Quem já desenvolveu em PHP sabe que a produtividade não era um requisito dos projetos. Hoje podemos de forma muito rápida, desenvolver uma aplicação Web arrastando e soltando componentes em um formulário. Isso no mundo PHP é uma revolução.

Para quem desenvolve em Delphi o aprendizado de Delphi for PHP será muito rápido e prazeroso, pois existem várias semelhanças com o Delphi 7. A semelhança está relacionada à tradução da VCL para PHP. Então se você sabe utilizar a VCL, saberá utilizar o Delphi for PHP.

Conclusão

Primeiramente quero deixar clara minha opinião quanto ao futuro do Delphi. Com todo o contexto histórico, apresentado nesse artigo, faço minha fundamentação para acreditar que ainda teremos um grande futuro para o Delphi.

Recebo vários e-mails questionando se vale a pena investir no aprendizado do Delphi e sou unânime em dizer que "sim", vale a pena. O Delphi é um produto sólido e que, em cada nova versão, é capaz de surpreender os usuários como novas facilidades e tecnologias. Na sua história é possível detectar isso como, por exemplo, o Delphi 3 que trouxe a possibilidade de desenvolvimento Web, onde muitos ainda não possuíam acesso à Internet. Outro exemplo é o Delphi 2, lançado no ano de 1995, que trouxe os componentes de cubos de decisão, que hoje são a bases para aplicações de BI (e pensar que em 1995 nem sabíamos o que seria BI).

A Borland sempre saiu na frente dos concorrentes e possui um grande diferencial: *"A boa relação com a comunidade de desenvolvedores"*. E essa relação é muito importante, pois, através dela podemos sugerir melhorias, correções de problemas e até mesmo novos produtos como foi o caso do Delphi for PHP. Acho que aí esta o ponto mais positivo da Borland/CodeGear.

Para finalizar, complemento o aspecto importante da escolha correta da versão e edição do Delphi a ser utilizada. Nessa decisão considere o Delphi 2007 que é o produto mais recomendado para o desenvolvimento de aplicações Win32. Considere as versões Turbo que são alternativas mais atraentes em questão custo/benefício. E, finalmente considere o Delphi for PHP para utilização em projetos que não exigem uma super infra-estrutura e investimentos muito altos.

Junto ao artigo, disponibilizei alguns arquivos utilizados na pesquisa bem como na seção *Links* temos blogs e vídeos do YouTube, que podem ajudar na escolha da melhor versão para seu caso, bem como contextualizar sua decisão. Os vídeos mostram os lançamentos de versões do Delphi.

Bem, agora você já pode fazer a escolha da sua versão! Não importa qual seja, mas, o que importa é que seja algum Delphi! ●

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma série de vídeo aulas de Adriano Santos, sobre as novidades do Delphi 2007.

www.devmedia.com.br/articles/listcomp.asp?txtsearch=Delphi+2007

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a vídeo aula de Fabrício Desbessel, que ensina como instalar o Delphi for PHP.

www.devmedia.com.br/articles/viewcomp.asp?comp=4889

Links

Artigo com Histórico do Delphi
www.thielke.com.br/alexandre/paradigma.htm

Blog do Nick Hodges
blogs.codegear.com/nickhodges

Blog do Andreano Lanusse
blogs.codegear.com/AndreanoLanusse

Blog do Marco Cantu (Autor das Bíblias do Delphi)
blog.marcocantu.com

Blog do David I
blogs.codegear.com/davidi

Vídeo de Lançamento do Delphi 1 (Speed is)
br.youtube.com/watch?v=ybGSzwJbjU4

Vídeo da Borland (inspirado em Jornada nas Estrelas)
br.youtube.com/watch?v=VmygOESWM5Q

Vídeo criado para a Borland Conference 1997 (insipirado em Star Wars). Muito legal, pois os maiores nomes da Borland são os atores.
br.youtube.com/watch?v=rLgk5CIf4XA

Túnel de Tempo da Borland (16 anos)
br.youtube.com/watch?v=VbWQybqAuOs

Vídeo para lembrar o espírito original da Borland (inspirado em Transformer)
br.youtube.com/watch?v=Aptgjeqg94o

Vídeo para lembrar o espírito original da Borland (Developer Tool Time)
br.youtube.com/watch?v=lMzD-7iGdHI

Vídeo para lembrar o espírito original da Borland (inspirado em Indiana Jones)
br.youtube.com/watch?v=-tIq4ICREkE

Vídeo do TurboMan
br.youtube.com/watch?v=zrj2kJbizBA

JavaScript

Incremente sua aplicação ASP.NET

Luciano Pimenta
(lucianopimenta@clubedelphi.net)
é Técnico em Processamento de Dados, Editor Técnico das revistas ClubeDelphi, .net Magazine, WebMobile e MrBool. subeditor de vídeo aulas do Portal DevMedia (www.devmedia.com.br). Palestrante da 4ª edição da Borland Conference (BorCon).

Mostrarei neste artigo algumas dicas de como usar JavaScript em aplicações ASP.NET, usando o Delphi 2006. Muitos desenvolvedores, ao iniciarem sua jornada no ASP.NET, querem que os projetos sejam semelhantes aos projetos desktop.

Com algum código JavaScript, podemos simular determinadas situações, mas deixo claro, que nem tudo é possível realizar no ASP.NET igual ao desktop, pois são plataformas diferentes (Web e Win32).

Neste artigo, veremos algumas dicas e técnicas para encapsular algumas chamadas de funções do *JavaScript* para usar em mais de um projeto ASP.NET.

Criando o projeto ASP.NET

Crie um projeto ASP.NET no Delphi 2006, dando o nome de "DicasJS". Veremos duas maneiras de usar o código JavaScript na aplicação, sendo uma no código ASPX outra no código Delphi.

Adicione dois botões na página da aplicação ASP.NET, colocando o *Text* como "ASPX" e "Delphi", respectivamente. No primeiro exemplo, vamos criar uma função no código ASPX para ser chamada pelo botão. Acesse o arquivo ASPX da página e adicione o código da **Listagem 1** após o elemento *<body>*.

> **Nota:** Tome cuidado, pois o JavaScript é *case-sensitive*.

Para exibir essa mensagem no botão ASPX, no *Page_Load* do formulário, adicione o seguinte código:

```
Button1.Attributes.Add('onclick',
  'javascript:AlertMessage()');
```

Para o segundo botão, vamos adicionar código no PAS. Use o código da **Listagem 2** no *Click* do botão.

No código anterior, declaramos uma variável do tipo *StringBuilder* e fizemos uma concatenação de *strings* criando uma função do *JavaScript*. Note que nesse

caso, não precisamos dar um nome para a mesma (se desejar criar na variável a mesma assinatura do método, como mostramos na **Listagem 1**, pode ser feito normalmente).

No final, verificamos se o *script* não foi registrado pela página e então, registramos o mesmo, mostrando a mensagem ao usuário. O valor do parâmetro do *IsClientScriptBlockRegistered* e *RegisterClientScriptBlock* é uma chave, portanto coloque o valor que desejar. Execute a aplicação e clique nos botões para exibir as mensagens (**Figura 1)**.

Você também pode colocar o código diretamente na propriedade *Attributes* do *Button*:

```
Button1.Attributes.Add('onclick',
  'alert(''Mensagem do JavaScript'');');
```

Objeto Window

Uma das técnicas mais procuradas pelos desenvolvedores é para abrir uma nova janela. Para isso, temos que usar o método *open* do objeto *window*, que possui vários parâmetros, sendo possível configurar menus, barras de botões, barra de status da nova janela.

No link *msdn2.microsoft.com/en-us/library/ms536651.aspx* temos a descrição de todos os parâmetros do *open*. Adicione um novo botão no formulário e crie uma nova página. No ASPX adicione mais uma função *JavaScript*, conforme o código da **Listagem 3**.

Além da página do nosso projeto (*WebForm2.aspx*), podemos passar uma URL para o parâmetro:

```
window.open("http://www.clubedelphi.net", '',
  'channelmode=yes, resizable=no, fullscreen=no,
  width=300, height=300');
```

Novamente no *Page_Load*, adicione o seguinte código, que vai chamar a função *JavaScript*:

```
Button3.Attributes.Add('onclick',
  'javascript:OpenWindow()');
```

Veja na **Figura 2** a aplicação com as janelas do projeto e com o site da ClubeDelphi.

As características dos parâmetros são:

- *resizable*: evita que a janela seja redimensionada;
- *fullscreen*: se a janela aparecerá no tamanho definido pelas propriedades *width* e *height*. Se *fullscreen* for igual a *yes* ou 1, os valores das propriedades são ignorados;
- *menubar*: exibe ou não a barra de menus;
- *scrollbars*: exibe ou não as barras verticais e horizontais;
- *status*: exibe ou não a barra de status (rodapé);
- *toolbar*: exibe ou não a barra de ferramentas (botões);
- *height* e *width*: tamanho da janela;
- *left* e *top*: posição da janela.

Janelas de consultas

Com as informações que temos até aqui, já podemos incrementar nossa aplicação ASP.NET, por exemplo, criando um tela de consulta, que será separada da janela da aplicação, semelhante a que temos em aplicações desktop. Vamos criar uma nova página e criar uma tela de cadastro semelhante à **Figura 3**.

Está fora do escopo do artigo as configurações de banco e componentes. Os fontes estarão disponíveis para download

Listagem 1. Criando um método JavaScript

```
<script language="javascript=">
function Message ()
{
  alert('Mensagem do JavaScript - ASPX');
}
</script>
```

Listagem 2. Usando mensagem JavaScript no PAS

```
uses System.Text;
...
var
  aScript: StringBuilder;
begin
  aScript := StringBuilder.Create;
  aScript.Append('<script language="JavaScript">');
  aScript.Append(
    'alert(''Mensagem do JavaScript - Delphi'');');
  aScript.Append('</script>');
  if not Page.IsClientScriptBlockRegistered(
    'client') then
    Page.RegisterClientScriptBlock('client',
      aScript.ToString);
end;
```

Listagem 3. Chamando o open do objeto window

```
function OpenWindow()
{
  window.open("WebForm2.aspx", '',
    'channelmode=yes, resizable=no, fullscreen=no, width=300, height=300');
}
```

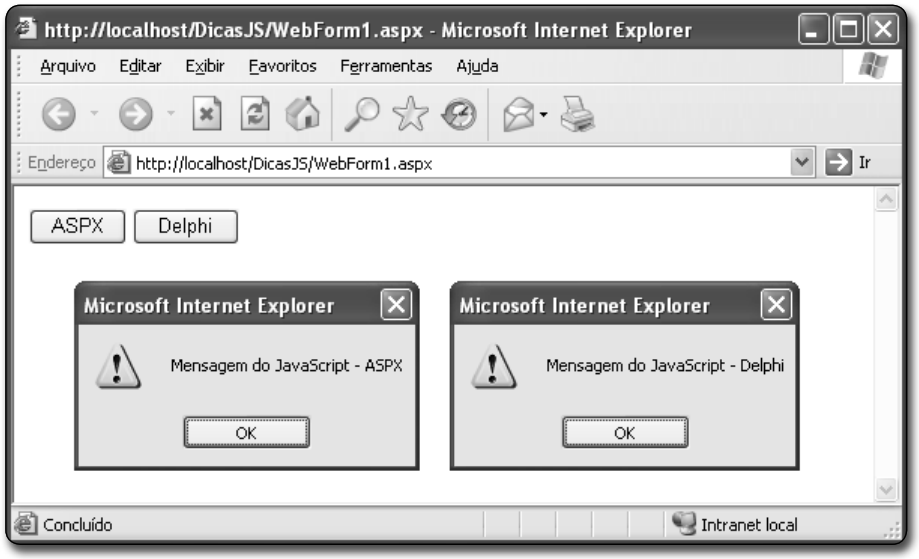


Figura 1. Mensagens do JavaScript no ASP.NET

e no site da ClubeDelphi temos vários artigos e aulas mostrando como conectar ao banco em aplicações ASP.NET.

O banco de dados foi criado para o artigo, possuindo duas tabelas apenas (CLIENTE e DEPENDENTE). A idéia é ter uma pesquisa no cadastro de Clientes e outras duas no cadastro de Dependentes (uma para o dependente e outra para o cliente).

Lembrando que a aplicação será semelhante a desktop na funcionalidade de trazer apenas um registro por vez, daí a necessidade da pesquisa.

Crie uma nova página e altere seu nome para "pesquisacliente.aspx". Nessa página, vamos adicionar alguns componentes, como podemos ver na **Figura 4**, responsáveis pela entrada dos dados de pesquisa e exibição do resultado.

O campo *Selecionar* do *DataGrid* é do tipo *Command>Select*. No *Localizar* do cadastro de clientes, vamos utilizar novamente *JavaScript* para chamar a tela de pesquisa, usando o código da **Listagem 4**.

Não esqueça de adicionar no *uses* o namespace *System.Text*. Lembre-se que o navegador pode bloquear a página pensando ser um *popup*. Você pode incrementar a tela de pesquisa, paginando o *DataGrid*, não esquecendo que nesse caso, a consulta deverá usar um *DataSet*.

Retornando o registro escolhido

OK, tudo legal, mas como retornar o registro que escolhi na tela de consulta? Como você verá no código-fonte, armazeno o código da tabela (ID_CLIENTE) em uma variável de sessão (poderia ser uma *QueryString*, *cookie* etc.), assim, basta armazenar na mesma o código escolhido no *DataGrid*.

Ao clicar no registro escolhido, quero que a janela seja fechada e o item atualizado no cadastro. Nesse caso, voltamos a usar *JavaScript* chamando o *close* do *window*, com o código da **Listagem 5** no evento *ItemCommand* do *DataGrid*.

Primeiro, preenchemos uma variável de sessão com o código do cliente, escolhido no *DataGrid*. Após, o *window.opener.location* faz um redirecionamento para a página de cadastro e o *window.close* fecha a janela.

Para entender melhor, no formulário de cadastro, no *Page_Load* é verificado se a variável de seção está preenchida, então é

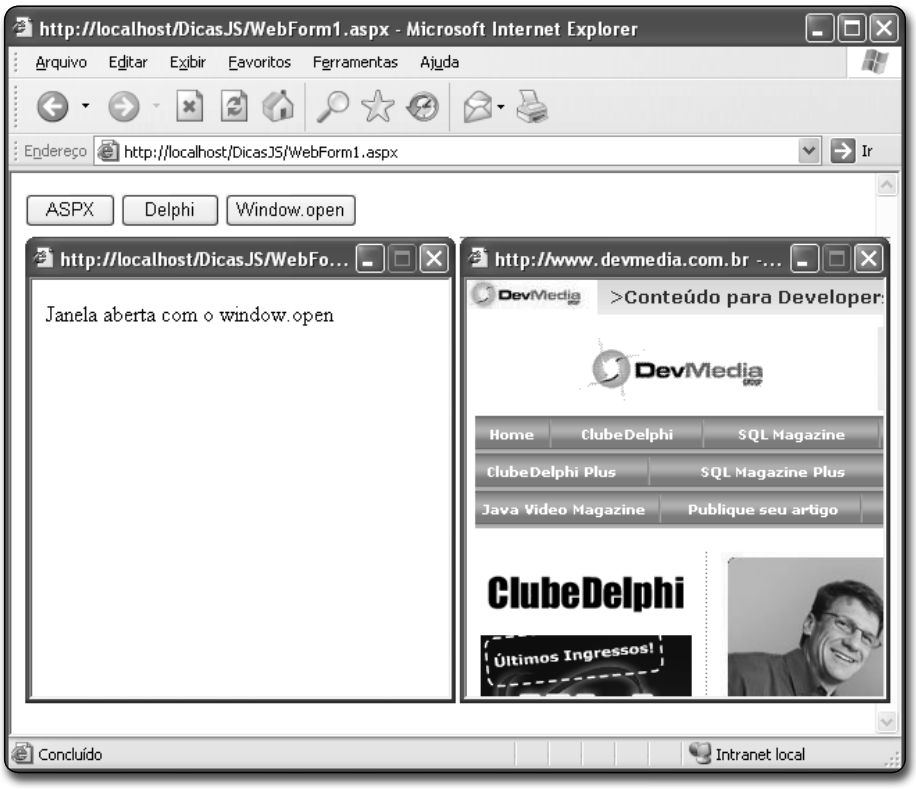


Figura 2. Abrindo páginas do projeto e sites

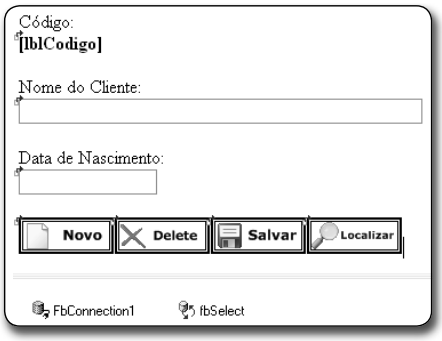


Figura 3. Tela de cadastro de exemplo

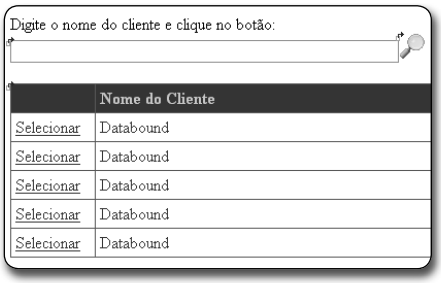


Figura 4. Tela de pesquisa do cliente

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp
Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Guinther Pauli, mostrando como acessar o Firebird em uma aplicação ASP.NET.
http://www.devmedia.com.br/articles/viewcomp.asp?comp=561

Listagem 4. Chamando a página de pesquisa de clientes

```
var
  aScript: StringBuilder;
begin
  aScript := StringBuilder.Create;
  aScript.Append('<script language="JavaScript">');
  aScript.Append('window.open("pesquisacliente.aspx",');
  aScript.Append('"", "channelmode=yes, ');
  aScript.Append('resizable=no,fullscreen=0, ');
  aScript.Append('width=500, height=500")');
  aScript.Append('</script>');
  if not Page.IsClientScriptBlockRegistered(
    'client') then
    Page.RegisterClientScriptBlock('client',
      aScript.ToString);
end;
```

feita a consulta no banco com o código da mesma. Veja o resultado na **Figura 5**.

Preenchendo controles

Depois da pesquisa, poderíamos por exemplo, preencher um *TextBox* com o nome do usuário escolhido no *DataGrid*. Por exemplo, no cadastro de dependentes temos uma consulta para o Cliente (o Dependente tem um cliente vinculado). Então, veja na **Figura 6** um exemplo do cadastro de dependentes.

Você deve estar se perguntando, vou ter de fazer outra tela de pesquisa? Se não, como fazer para saber quando é o cadastro de clientes e quando é o de dependentes? Simples, use uma *QueryString*, variável de sessão ou *cookie* para indicar de onde a tela de pesquisa foi chamada.

Vamos preencher o *TextBox* com o nome do cliente escolhido na tela de pesquisa. Para isso, devemos chamar a tela de pesquisa no cadastro de dependentes usando o código a seguir:

```
...
aScript.Append(
  'window.open("pesquisacliente.
   aspx?act=dep",');
aScript.Append('"", "channelmode=yes, ');
aScript.Append('resizable=no,fullscreen=0, ');
```

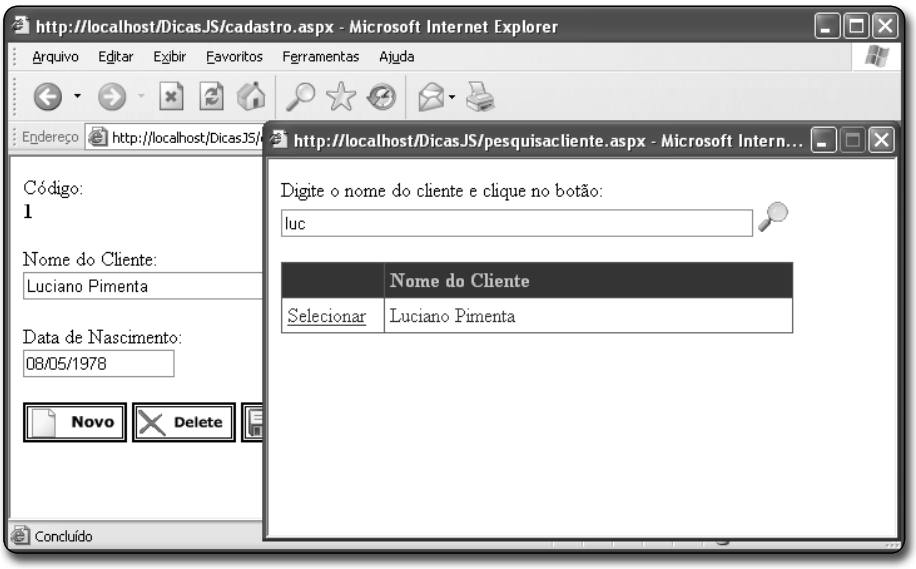


Figura 5. Fazendo a busca do cadastro

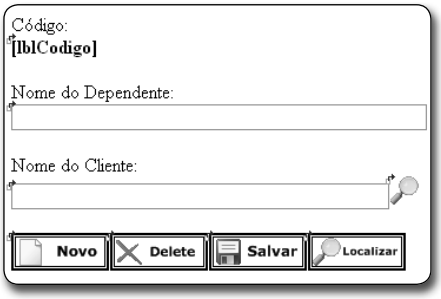


Figura 6. Cadastro de dependentes

Fale conosco!

Conheça a Central de Atendimento on-line da DevMedia. Agora ficou muito mais fácil de acompanhar sua assinatura, alterar seus dados cadastrais e retirar qualquer dúvida sobre nossos serviços e produtos. Para isso basta ter em mãos seu login e senha.

Na Central de Atendimento você:

- Consulta as perguntas mais frequentes

- Renova por boleto ou cartão de crédito

- Consulta e altera seus dados cadastrais

- Consulta a data de envio de cada exemplar

DevMedia GROUP

Para entrar em contato com a DevMedia, basta postar sua dúvida na própria Central de Atendimento através de um sistema web, prático, seguro e fácil de usar. Com essa nova ferramenta você tem a garantia de receber suas respostas em menor tempo e com alto padrão de qualidade.

www.devmedia.com.br/central

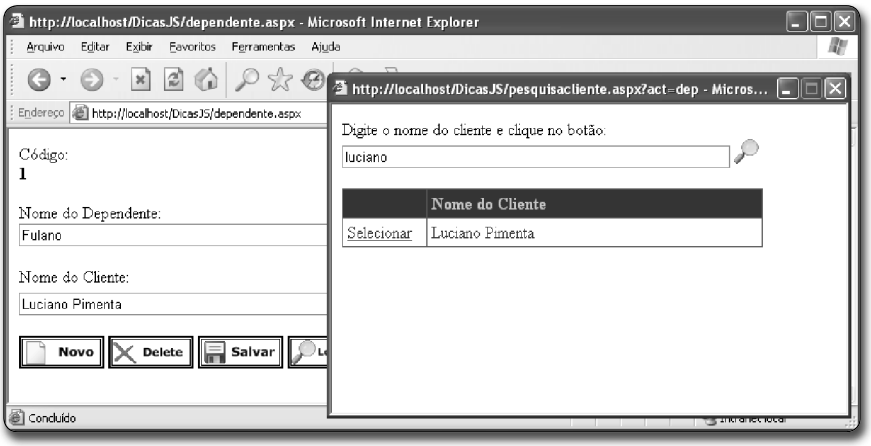


Figura 7. Preenchendo controles

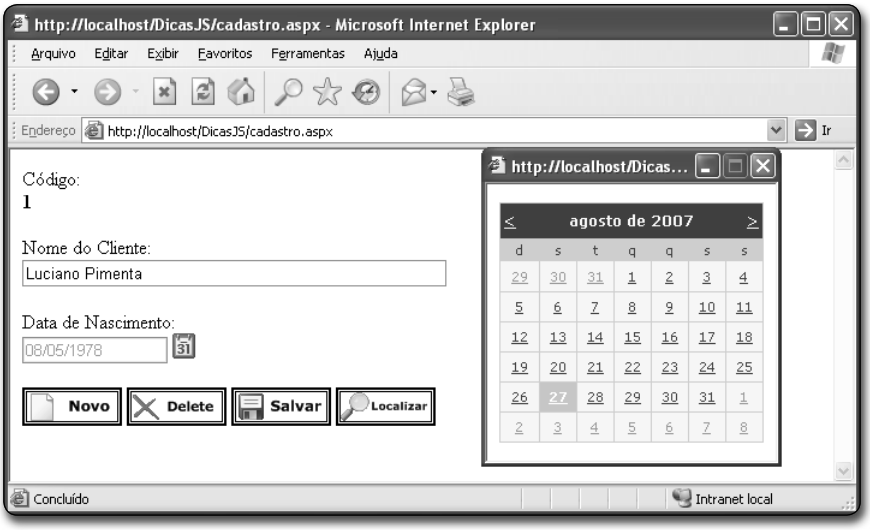


Figura 8. Escolhendo a data e desabilitando o controle

Listagem 5. Fechando a janela de consulta

```
var
  aScript: StringBuilder;
begin
  { Passa para a session o código do cliente }
  Session['ID'] := DataGrid1.DataKeys[e.Item.ItemIndex];
  { Se clicou no Selecionar, executa script }
  if e.CommandName = 'Select'  then
  begin
    aScript := StringBuilder.Create;
    aScript.Append('<script language="JavaScript">');
    aScript.Append('window.opener.location = ');
    aScript.Append('"cadastro.aspx";');
    aScript.Append('window.close()');
    aScript.Append('</script>');
    if not Page.IsClientScriptBlockRegistered('client') then
      Page.RegisterClientScriptBlock('client', aScript.ToString);
  end;
end;
```

Listagem 6. Verificando a página chamadora

```
...
aScript.Append('<script language="JavaScript">');
if Request.QueryString['act'] = 'dep' then
begin
  aScript.Append('window.opener.document.forms[0]');
  aScript.Append('["txtNomeCliente"].value ="' + e.Item.Cells[1].Text + '"; ');
end
else
begin
  aScript.Append('window.opener.location = ');
  aScript.Append('"cadastro.aspx";');
end;
aScript.Append('window.close()');
...
```

No *Selecionar* do *DataGrid*, devemos alterar o código para o da **Listagem 6**, que verifica o formulário que chamou (*QueryString* chamada *act*) a tela de pesquisa e executa o código de acordo com a opção.

Veja que modificamos a página que deve ser redirecionada e utilizamos a coleção de controles da página que chamou (forms[0]), passando o nome do controle que queremos preencher, nesse caso, o *txtNomeCliente*.

Note também que não redirecionamos para a página de dependentes, pois queremos apenas preencher o controle e não atualizar a página (isso deve ser feito na pesquisa de dependentes, como fizemos no de clientes). Atente para o nome correto do controle, em seguida, continuamos com o código normal.

Ao executar a aplicação basta localizar o cliente na mesma tela de pesquisas usada no cadastro, e ver que o valor escolhido pelo usuário está sendo preenchido no *TextBox* da página de cadastro (**Figura 7**).

Trabalhando com calendário

O exemplo com calendário é bastante parecido com o exemplo anterior, mas o que diferencia, é que podemos formatar a data de retorno e não precisamos validar a data do usuário. Adicionei um *ImageButton* ao lado do campo *Data de Nascimento* no cadastro de clientes.

Criei uma nova página com um calendário formatado. O código para chamar o formulário é o mesmo usado para a pesquisa. No calendário, vamos usar o código da **Listagem 7**, no evento *SelectionChanged* do *Calendar*.

Se preferir, pode colocar o código anterior em um botão. Formatamos a data escolhida através do *System.String.Format*, utilizando o formato *dd/MM/yyyy*. Agora, você tem um calendário para que o usuário escolha a data, sem precisar validar a mesma (basta desabilitar o controle para que o usuário não digite a data escolhida), como podemos ver na **Figura 8**.

Customizações

Colocar o código *JavaScript* no Delphi pode ser um pouco cansativo, então, o que acha de customizarmos alguns métodos? Para executar um determinado bloco de código, chamar o método de

A EDIÇÃO QUE VOCÊ PRECISA ESTÁ ESGOTADA?

SEUS PROBLEMAS ACABARAM!!!

Seja um assinante Gold!

Com a assinatura Gold você já pode consultar online
todos os artigos publicados na sua revista desde a edição nº 1.

DevMedia GROUP

Saiba Mais! Acesse:
www.devmedia.com.br/assgold

Para mais informações:
www.devmedia.com.br/central

Assinatura Gold

Atenção: Já encontram-se disponíveis as assinaturas GOLD das revistas WebMobile e .net Magazine.

caixa de mensagem, abrir uma janela com *open* e modal.

Veja no código da **Listagem 8** a customização de alguns métodos para que possamos utilizá-los de uma maneira mais fácil na aplicação, usando uma *unit* separada, por exemplo.

Veja que poupamos bastante código ao customizar os métodos, por exemplo, no *OpenWindow* passamos a página a ser exibida, bem como o tamanho da mesma. Veja na **Listagem 9**, exemplos de utilização dos métodos criados.

Conclusão

Mostrei neste artigo, algumas dicas de *JavaScript* para serem adicionadas em aplicações ASP.NET com Delphi. Vale lembrar que alguns navegadores podem ter incompatibilidade com o código. Os testes realizados utilizaram somente o Internet Explorer 6.0.

Portanto, é válido testar a sua aplicação em outros navegadores, antes de colocar a mesma para a utilização dos usuários do site. Um grande abraço a todos e até a próxima! ●

Listagem 7. Evento SelectionChanged do Calendar

```
var
  aData: string;
  aScript: StringBuilder;
begin
  aData := System.&String.Format('{0:dd/MM/yyyy}', Calendar1.SelectedDate);
  aScript := StringBuilder.Create;
  aScript.Append('<script language="JavaScript">');
  aScript.Append('window.opener.document.forms[0]');
  aScript.Append('["txtDataNascimento"].value="'+aData + '"; window.close();');
  aScript.Append('</script>');
  if not Page.IsClientScriptBlockRegistered('client') then
    Page.RegisterClientScriptBlock('client', aScript.ToString);
end;
```

Listagem 8. Customização dos métodos

```
procedure OpenWindow (aPage: System.Web.UI.Page;
  aRedirect, aWidth, aHeight: string);
var
  aScript: StringBuilder;
begin
  { abre uma nova janela do browser }
  aScript := StringBuilder.Create;
  aScript.Append('<script language="JavaScript">');
  aScript.Append('window.open("'+ aRedirect + '", ');
  aScript.Append('"", "resizable=no, menubar=no, ');
  aScript.Append('scrollbars=yes, status=yes, ');
  aScript.Append('left=350, top=150, width='+
    aWidth + ', height=' + aHeight + '")');
  aScript.Append('</script>');
  if not aPage.IsClientScriptBlockRegistered(
    'client') then
    aPage.RegisterClientScriptBlock('client',
      aScript.ToString);
end;

procedure ExecuteScript (aPage: System.Web.UI.Page;
  aScript: string);
var
  Script: StringBuilder;
begin
  { Execute script passado como parâmetro }
  Script := StringBuilder.Create;
  Script.Append('<script language="JavaScript">');
  Script.Append(aScript);
  Script.Append('</script>');
  if not aPage.IsClientScriptBlockRegistered(
    'client') then
    aPage.RegisterClientScriptBlock('client',
      Script.ToString);
end;

procedure MessageScript(aPage: System.Web.UI.Page;
  aTexto: string);
var
  Script: StringBuilder;
begin
  { mensagem com JavaScript }
  Script := StringBuilder.Create;
  Script.Append('<script language="JavaScript">');
  Script.Append('alert("'+ aTexto + '");');
  Script.Append('</script>');
  if not aPage.IsClientScriptBlockRegistered(
    'client') then
    aPage.RegisterClientScriptBlock('client',
      Script.ToString);
end;
```

Listagem 9. Exemplos de chamadas dos métodos

```
Mostra a caixa de mensagem
if Update then
  MessageScript(Page,
    'Atualização realizada com sucesso!');
Executa o script passado como parâmetro
ExecuteScript(Page, 'window.opener.document.forms[0]'+
  '["txtData"].value="'+ aData + '"; window.close();');
Abre a janela de localizar clientes
OpenWindow(Page, 'pesquisacliente.aspx',
  '600', '450');
```

DevMedia
GROUP

O conteúdo ao seu alcance!

+ de **600 artigos** impressos por ano

+ de **1500 vídeo-aulas** de acesso imediato

+ de **24 cursos** online

+ de **130 vídeos inéditos** todos os meses

Se você busca **conteúdo** de qualidade e aprendizagem rápida.

Aqui é seu lugar!

Acesse e confira:

www.devmedia.com.br

Cursos Online

Assinatura

ClubeDelphi PLUS

Mais conteúdo DELPHI por muito menos!

A Revista **ClubeDelphi** oferece para seus assinantes uma série de **Cursos Online** de alto padrão de qualidade . Conheça abaixo os cursos já disponíveis..

Curso em destaque

Aplicações Client/Server com dbExpress e Firebird

Confira neste curso online como criar uma aplicação client/server completa no Delphi 7, utilizando dbExpress e Firebird.
Aprenda também: Como trabalhar com um driver dbExpress específico e gratuito para o Firebird ; Como utilizar Orientação a Objetos, e técnicas como herança visual de formulários e relatórios ; saiba como colocar regras de negócios no banco de dados, usando Triggers e Stored Procedures; E crie relatórios com o Quick Report, Rave Reports e Report Builder, e muito mais.

Confira o plano de aula completo:
www.devmedia.com.br/clienteserver

A sua melhor opção de aprendizagem!
Assine a **ClubeDelphi** e comece já seu treinamento!
www.devmedia.com.br/assine

Outros cursos disponíveis: **www.devmedia.com.br/curso**

- Curso Online - Sistema SysCash
- Aplicações client/server com Windows Forms no Delphi 2006
- Criando uma Aplicação multi-camadas Completa com Delphi
- Aplicações WEB com IntraWeb e Delphi 7 (Delphi Win32)

DevMedia GROUP **Mais Informações: www.devmedia.com.br/central - Tel.: 21 2220-5375 / 2220-5435**